Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

A Meta-Approach to Guide Architectural Refactoring from Monolithic Applications to Microservices

Qiwen Gu

Course of Study:	INFOTECH
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Jonas Fritzsch
Commenced:	April 30, 2020
Completed:	December 03, 2020

Abstract

The concept of microservices in the software development industry is getting growing attention nowadays. This architectural style is widely discussed both in industry and academia. Refactoring a monolithic application into a microservice application is common practice. Nevertheless, software architects and developers often find it difficult because they lack a structured overview of various migration approaches. Even though literature views about microservice migration were conducted [33][83], they were either obsolete or did not follow a systematic approach to ensure correctness and reproducibility of results. The goal of this study is to provide a classification framework as well as a web-based tool that can guide software architects and developers to comprehend up-to-date migration approaches and select a suitable one according to their requirements. In order to achieve this, a systematic literature review was conducted, resulting in thirty-one contributions from 2017 to 2020. Next, a web-based tool was developed based on the knowledge repository created after review. An evaluation of the developed tool by experts and students in the field revealed that it was able to serve the predefined purpose. The proposed framework, as well as the web-based tool, can provide the users a comprehensive overview of microservice migration and various practical approaches.

Keywords: Microservices, Monolith Migration, Architectural Refactoring, Microservice Migration Framework

Contents

List of Figures

List of Tables

List of Codes

Chapter 1

Introduction

The concept of microservices in the software development industry is getting growing attention nowadays. This architectural style is widely discussed in many contributions. The microservice architecture consists of several light-weight, single-responsibility, and self-contained services that interact with each other to fulfill the business functionalities desired by users [92][108][103]. It has many advantages, for instance, the service can be developed in various languages with different tools which best fits the user's requirements; each service can be scaled easily and deployed onto independent platforms; the development teams can update and test the services separately, so the system is easily manageable, and failures can be enclosed within the service border, ensuring fault tolerance and reliability [98]. When the legacy applications fail to meet the performance requirements by development teams, or when development teams meet technical problems during development, deployment, or maintenance, software architects tend to seek a nice solution. Often a microservice architecture seems to be an attractive one because of its fancy concept and trend [99][50]. It is a common approach to migrate the legacy application into a new microservice environment. Many internet companies such as Amazon, Netflix, Google, Alibaba, etc. [87] have already adopted microservice architecture. Also, many modern cloud-enabled platforms, including Google Cloud, and Amazon AWS, also support microservice applications to exploit the benefits. However, to some software such as enterprise systems, this architecture is not dominant yet [25].

Even though various contributions have discussed migration approaches or frameworks, a clear guidance on architectural refactoring of legacy application remained unclear to many researchers because they are facing with different environments, and they have different objectives. There are many ways to decompose and migrate the monolithic application to microservices, and they usually vary significantly regarding quality metrics, practical profoundness, applied techniques, tool support, input resources, validation processes, and so on. Considering this fact, it brings challenges to software architects and developers to have a general understanding of available approaches or frameworks and their capabilities. It may also be difficult and time-consuming for them to choose a suitable way to migrate their own monolithic applications in proper granularity. Moreover, the number of researches and contributions in academia and industry grow continuously, and they keep updating the current status of the study. This fact also makes previous studies [33][84] to be obsolete quickly.

The goal of this study is to create a structured overview of up-to-date microservice architectural refactoring approaches and frameworks. So it can aid software architects and developers to check and select appropriate migration strategies and refactoring approaches that are applicable for their specific system.

In order to achieve this goal, a systematic literature review was performed by referring to a guideline proposed by Kitchenham and Charters [52]. They introduced a set of explicit and formal processes to conduct the literature review. The research protocol was defined at first, and it was strictly executed during the literature review to ensure that the achieved result was correct and reproducible. In this review, I included new contributions that had not been captured by previous meta-studies yet. I focused mainly on the underlying techniques used for application

decomposition. In addition, intentions for migration and evaluated quality metrics of microservice applications were also further investigated. Based on the knowledge acquired from the literature review, a comprehensive classification framework for refactoring approaches was designed. Next, a web-based tool was conceptualized and implemented based on an extensible repository that contained all the data of analyzed migration approaches collected from the literature review. According to users' requirements, this tool can provide users appropriate contributions to guide the choice of suitable refactoring techniques and approaches. Finally, a formal evaluation of the tool was done among consulting experts and students in the relevant field. According to their feedback, I could tell whether the tool was helpful for partial use or not.

According to the guideline by Kitchenham and Charters [52], research questions should be specified because they will drive the whole review methodology [52]. Based on my goal and the previous research gaps, the research questions were formulated as follows:

RQ 1: What approaches or frameworks for migration scenarios are proposed in the scientific literature?

Especially, the RQ 1 can be further divided into two sub-questions:

RQ 1.1: How can the proposed approaches or frameworks be classified? What strategies and techniques did they implement?

RQ 1.2: What are relevant intentions and quality metrics in a microservice migration/ refactoring scenario?

RQ 2: How can we design a tool to serve the architects and developers as a guidance for microservice migration?

These two questions were particularly inspected in this study, and they will be answered based on the evidence and knowledge acquired during the systematic literature review and web-based tool design.

The remainder of this thesis is organized as follows: Section 2 presents the background and related work. In Section 3, I will introduce the basic methodologies I have applied for the systematic literature review. In Section 4, the detailed result and knowledge gathered from the literature review will be described. Then Section 5 will present the design, implementation, and evaluation of the corresponding web-based tool. In Section 6, the result of the whole study will be discussed and interpreted. In Section 7, I will point out potential threats to the validity of this work. Finally, the conclusions and suggestions for future work will be stated in Section 8. In addition, the appendix section at last will provided some raw materials and data generated during my thesis.

Chapter 2

Background and Related Works

In this chapter, the background knowledge of this topic, as well as previous related researches, will be mentioned. What's more, some specific architectural design concepts, including Domain-Driven Design and Model-Driven Design, will also be introduced.

2.1 Background

In order to provide readers a basic understanding of the microservice architecture and its characteristics, a brief introduction of microservices is given below. In addition, the concept of the traditional monolithic application is also discussed so that a comparison between these two architectures is obvious to readers.

2.1.1 Monolithic Application

The monolithic application has a traditional architecture, which means that the application is developed and deployed as a self-contained entity having all responsibilities. Since the whole application is implemented within a single package and homogeneous environment (same framework and language), it is easy to develop and test [91][50]. The characteristic of monolithic architecture is tight coupling, which means that its components heavily rely on each other to conduct a task [50]. To achieve scalability, a load balancer is used to run multiple application copies when the application encounters performance bottleneck [87].

The benefits of monolithic application can be listed as follows:

(1) It requires less effort and operational overhead because only one application is developed and deployed. All components are running within a single entity [50].

(2) This is good for small projects to adopt monolithic architecture because it is simple [91].

(3) It is developed using the homogeneous technique and deployed in one particular environment, so it is also easy to maintain.

However, drawbacks also tend to appear and become outweigh the benefits [69] when the application becomes large and complex after a few years:

(1) The monolithic architectural style is difficult to change due to tight coupling components and legacy techniques. When the application faces bugs, bottlenecks or requirements for additional functionalities, the change of any part requires testing and redeploying the entire application [92]. Unexpected errors may appear after updates, and this brings difficulty to

maintenance when the application is large and cumbersome. Plenty of efforts and time are required for the updates. Besides, it is not suitable for continuous deployment [50].

(2) It is also hard to scale a monolithic application because of the same reason. The system has no choice but to run multiple application copies in parallel to resolve performance bottleneck. This wastes unnecessary resources and adds additional cost [87][65]. However, when different components have conflicting resource requirements, it required more effort to orchestrate them carefully [92][65].

(3) The reliability of the whole application may also be influenced even if only one component breaks down [50].

(4) Because of high coupling and legacy codes, it is challenging to adopt new languages and frameworks. This brings potential risk of technology lock-in [91].

When the maintenance and further improvement of a monolithic application become costly in time and effort, the development teams may think about adopting a new architectural solution. One possible answer for overcoming such limitations is microservice architecture [91].

2.1.2 Microservice Architecture

Microservice was first introduced in 2011. A growing interest in the software industry can be observed because of its benefits over monolithic applications [25]. It can be defined in various ways. One of the most popular definitions is introduced by Sayara et al. [92], that is, microservice architecture evolves from service-oriented architecture (SOA). SOA has separate components running in parallel, making the application loosely coupled and high cohesive. However, it also has shared service contracts, shared databases, etc., which still contains correlation problems.

Microservice architecture improves further by minimizing or totally separating these shared resources, so that the whole application consists of light-weight and well-defined services that are independently developed and deployed using different tools, languages, platforms, etc. [92][108][103]. Each service should be assigned with only one business responsibility ideally, and they interact with each other to fulfill tasks by means of Inter-Process Communication (IPC) mechanism using interface such as REST API [108][91]. Typically, Domain-Driven Design (DDD), Single Responsibility Principle (SRP), or Conway's Law are applicable to design microservices according to business functionalities [90]. And it is a common practice to design the microservice application based on an existing monolithic one.

Microservice application is naturally scalable. It is usually integrated using service discovery, API gateway, and circuit breaker. They are used for service registration, client-service communication, and error handling [20]. Therefore, the services can be scaled separately by deploying service copies on different hosts. During system building, automation techniques such as continuous integration or continuous delivery are often applied.

One challenge of microservice is to define the number and size of each service properly. According to the single responsibility principle [90], a service should be responsible for one business functionality, which will influence the size of the service. However, the granularity of such business functionality is unclear and depends on actual scenarios [92].

Another challenge is to ensure the performance of the overall application because communications between services across the network boundaries may decrease performance [46]. So, the coupling of services, which is represented by the number of function calls between services, should be paid special attention and designed carefully.

To sum up, these challenges require the developers and architects to partition the microservice application properly. Such design in practice is often intuitive and based on practical experience

[103]. Considering the fact that most microservice applications are developed based on legacy applications, a good separation of business functionalities is therefore essential.

The advantages of microservices include:

(1) It is easier to realize high-concurrency and high-capacity [87].

(2) It is suitable for large project teams and complex applications which require frequent updates and maintenance [87]. Only updated services required redeployment without restarting the whole application. And maintenance focusing on one specific service is also easy [91].

(3) It is more scalable than monolithic application because each microservice can be scaled independently.

(4) Faults can be isolated within service boundaries because of loose coupling and high cohesion [103]. Therefore, the reliability of the whole application can be ensured.

(5) The independent services own private resources and databases. Therefore, performance can be enhanced separately [103].

(6) Each microservice can use different program languages, frameworks, libraries, and other resources flexibly [50].

(7) Because of the characteristics of microservice architecture, it matches for cloud deployment [50]. Besides, it is commonly integrated with DevOps and agile development. The automation techniques are applied to fasten the development and deployment process [20].

(8) Microservices architecture usually partition services according to business functionalities, so the application can be better aligned with business needs, and developers can better understand users' requirement [50].

In spite of its advantages, there still exist some limitations:

(1) The microservice architecture is more complicated than monolithic application because of the separation of services. In other words, it can be treated as a distributed system [50].

(2) Inappropriate service partition can be costly [103]. For instance, if an application is partitioned wrongly, the services are coupled to some extent, and they will generate many inter-service communications via the network, which will harm the performance [50].

(3) The implementation of such architecture lacks explicit methodologies and is heavily based on intuitive experience.

Please note that microservice architecture is not always the best answer to architectural design. Kazanavičius and Mažeika [50] insisted that: "Simply turn to microservice architecture because of its fancy idea and growing popularity is a bad idea." Software architects and developers should choose the form of structure freely based on their own situation and evaluation.

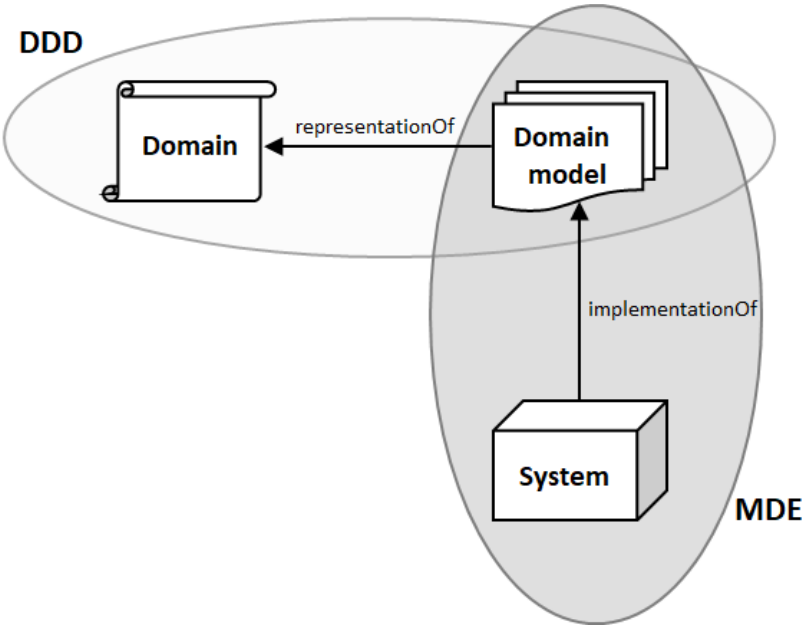


Figure 2.1: Domain-Driven Design (DDD) and Model-Driven Design (MDD) [15]

2.1.3 Architectural Design Concepts

Additionally, there were two formal design patterns which were mentioned in previous research: Domain-Driven Design (DDD) and Model-Driven Design (DDD). They can be used during microservice partitioning. The basic concept and comparison will be introduced in the following paragraph.

Domain-Driven Design (DDD)

Eric Evans [30] defined Domain-Driven Design in his book as "the concept that application structure and source code, including class names, class methods, and class variables, should be consistent with the business domain. It aims at core business domain and corresponding business functionalities, and focuses on iteratively designing a conceptual model based on a particular domain together with technical and domain experts". Evans explained several terminologies in order to create a precise mapping of knowledge from concepts to tangible examples [30].

(1) "Domain" (in software engineering) indicates a specific area where the user applies a program.

(2) "Context" explains the meaning a word or statement, which is used during the definition of a domain.

(3) "Model" represents the selected characteristics of a domain in abstraction, and it is usually used to solve the corresponding problems.

(4) "Bounded Context" is used when a software project defines multiple models. It is used to clarify the context within which a model applies, set boundaries between development teams in terms of responsibility, code repository, resources and so on. It makes sure that the models are consistent within these boundaries.

(5) "Context Map" solves the problems caused by "Bounded Context", that is, isolation and unawareness of other bounded contexts due to the boundaries, and the absence of a global view. The problems will make the edges vague again. Therefore, the "Context Map" uses ubiquitous language among different development teams to define models' names and describe explicit points of interaction between them. It will create a comprehensive overview of the whole domain area.

Model-Driven Design (MDD)

Besides Domain-Driven Design, another pattern called Model-Driven Design is also frequently used during the migration process. Jordi Cabot states that Model-Driven Design shares many similar aspects with Domain-Driven Design. For instance, both of them model the problem domain of a system and focus on platform-independent solutions during the design stage. He shows Figure 2.1 to illustrate the concept and comparison of Domain-Driven Design (DDD) and Model-Driven Design (MDD). As we can see, Model-Driven Design is a framework that realizes the concept of Domain-Driven Design by model transformation and code generation techniques. The domain models can be further applied to generate a software system for model management. Hence, Model-Driven Design benefits developers more in the development process, it can provide assistance such as introducing domain-specific languages for communication between team members [15].

After a brief introduction about background knowledge and previous academic literature, the basic concept of microservice, as well as the research status about microservice migration before 2017, were studied. Therefore, we can start to perform a systematic literature review to continue the study and fill the research gaps.

2.2 Existing Migration Approaches

In order to know the current status of research about this topic, two contributions were reviewed before the actual start of this study.

According to the study by Fritzsch et al. [33], an overview of decomposition approaches with regard to ten contributions was provided in their research. The overview had a relatively clear and detailed classification describing the characteristics of each approach. As an illustration, they classified the contributions according to technique type, applicability, process strategy, atomic unit/ granularity, required inputs, expected output, result evaluation type, tool support, and validation steps.

(1) Technique Type
Fritzsch et al. clarified the terminologies used in their contribution [33]. In particular, technique type groups specific analysis methods into four main categories: Static Code Analysis Aided (SCA) method, Meta-Data Aided (MDA) method, Workload-Data Aided (WDA) method, and Dynamic Microservice Composition (DMC) method. SCA method decomposes the applications by inspecting their source code; MDA method takes advantage of more conceptual resources, notably, architectural documents such as UML diagrams, use case descriptions, etc.; WDA method makes use of the applications operational data like log files about specific modules or functions because these data indicate the performance or communication intensity. By analyzing them, suitable service cuts (granularity) can be derived; DMC method is similar to WDA one, but it focuses more on the iterative improvement of the service cuts according to run-time workload and environment. Software developers can combine multiple techniques during the migration process in consideration of the structure and available resources of the application.

(2) Process Strategy
The next aspect is process strategy. As we can see in Figure 2.2, one microservice migration

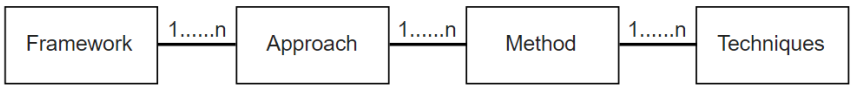


Figure 2.2: Relationship Among Each Level

framework may include several approaches. They usually express comprehensive ideas about migration measures to be taken and summarize them into a brief description. And in one approach, the developers can exploit multiple methods as migration process strategies. They are systematic procedures which denote one or more concrete techniques, that is to say, what should developers do to migrate the monolithic application. Subsequently, the lowest level is techniques, which are the most specific and practical ways to implement the migration and development procedures, in other words, how should developers realize the methods.

(3) Applicability
Regarding applicability, it describes the scenarios of microservice implementation. The three basic scenarios are greenfield development (GR), monolith migration (MO), in other words, brownfield development, and one that is applicable for both cases (GRMO). John Wade [105] has compared the concepts of greenfield development and brownfield development. He claimed that monolith-migration means developing microservices based on legacy applications, such as decoupling different modules or adapting and improving old source code to new applications. This is a conservative approach that relies on the existing feasible techniques. Therefore, it is less likely to fail again. Also, the developers can spend less effort and time during migration because they have already worked with these environments for years. However, risks still exist when it comes to performance bottlenecks, maintenance cost, cumbersome source codes and so on. Legacy codes may be difficult to change on account of highly coupled codes, old technologies, unexpected bugs caused by changes, etc. On the other hand, greenfield development means that the microservice application is developed from draft. Therefore, the business functionalities remain similar, but the internal software architecture and applied technologies (programming language, platform, etc.) may be utterly different. In that case, more suitable technologies, tools, environment, and structure can be applied during the development process, and that brings advantages such as better performance and legacy technical restrictions resolve. Nevertheless, this scenario also has disadvantages, including higher risk of failure, longer learning curve for new technologies and so on. In addition, some approaches are applicable for both greenfield and brownfield development, they usually provide generic decomposed structures of microservices, and architects can make trade-offs and choice a development scenario which best fits their requirements.

(4) Granularity
After this, Fritzsch brought up the concept of atomic unit/ granularity. He explained it as "the smallest entity that the microservice application will deal with" [33]. This term is highly affected by the decomposition strategies, and in return it will influence the extent of coupling and cohesion among different entities. This may also affect the final performance. To illustrate, the fine-grained decomposition will create a considerable amount of highly cohesive microservice entities. They can work in parallel and improve performance but also create managing and scheduling challenges due to their complex relationship and frequent intercommunication [18]. On the other hand, coarse-grained decomposition will result in several relatively big entities, so less communication overhead is possible, but the workload may be imbalanced and performance bottleneck will appear after these services scale up. Another

problem is maintainability, considering the fact that each entity's size may be hundreds or even thousands of lines of code, which makes them difficult to change and more errors may show up if developers change a single line of code [18]. Overall, granularity needs to be conscientiously decided according to actual functional and non-functional requirements. There is no single correct answer. Some approaches will calculate the granularity for reference, while others will leave this decision to development teams.

(5) Required Inputs and Expected Output
Moreover, required inputs and expected output are also informative aspects to development teams. These aspects inform them about necessary artifacts (documents or resources) needed for the proposed approach. In particular, one approach may require UML diagrams and original source code for application analysis and decomposition. The output aspect can show possible outcomes after the migration process, such as s suggested list of microservices or decomposed clusters with their relationships. Architects can refer to this information and choose suitable approaches according to their own application.

(6) Evaluation type
The next classification aspect is result evaluation type, and this illustrates what quality attributes have been measured during the evaluation stage. The quality attributes are quantifiable metrics, which reflect the quality and performance of the new microservice application. Fritzsch gathered these attributes for each contribution. For instance, some mentioned response time, team size, average domain redundancy as evaluation metrics [67][72]. Later, experiments, examples, or case studies were carried out during the validation stage, and these metrics will be utilized. More specifically, a contribution may introduce a monolithic legacy application for migration, recording its size in LOC, number of classes and methods, team size, year of development, functionalities, etc. Next, it will document the steps which follow the proposed approach and migrate the application. And then it measures the quality attributes to check whether the new microservice application can meet the expected requirements and whether this approach is feasible in reality. In addition, it may discuss the advantages or disadvantages and suggest further study and improvement of this approach. The aim of these two stages is not only to prove the effectiveness of a specific approach but also to demonstrate its applicability scenario under industrial or practical development environments. Therefore concrete and comprehensive argumentation is available to architects and enables them to judge the suitability of the approach to their project. They can also evaluate the quality of contributions, whether they are mature approaches or just conceptual ideas without practical validation which require further studies.

The literature review conducted by Fritzsch and others [33] is detailedly introduced because they provided some explicit aspects to classify the migration approach and techniques. This knowledge acted as a basis for my study. And I can further extend these aspects to generate a more comprehensive overview of microservice mitigation.

Additionally, Ponce, Márquez, and Astudillo conducted a rapid review about microservice migration [84]. They classified the migration strategies into Model-Driven Design (similar to Domain-Driven Design), Static Analysis, and Dynamic Analysis (similar as workload-data aided). They found that most approaches used Model-Driven Design or Static Analysis during the migration process. Moreover, they identified that seventy percent of migrations happened on web-based systems, and ninety percent of the microservice applications were developed using object-oriented programming languages. Nearly half of contributions validated the proposed approaches by case studies, following by experiments and then examples. They claimed that the challenges for migration included database migration, business capabilities partitioning, resource management, environment setting and so on.

2.3 Related Work

Fritzsch et al.conducted another research [34] to study the intentions, strategies, and challenges for microservice migration in the industry. They interviewed ten participants who had at least five years of working experience. They identified that maintainability, analysability, traceability, modifiability, flexibility, performance improvements, shorter time to market were major intentions for migration. With respect to migration strategies, most development teams applied rewrite or strangler patterns. In addition, they also identified several challenges, including finding the right service cut, adopting DevOps or agile pattern for software development and integration, team collaboration and human resource management.

Similarly, Taibi, Lenarduzzi, and Pahl [99] pointed out that maintainability, scalability and delegation of team responsibilities were three main drivers for migration. This matched the observed benefits of migration such as improvement of maintainability and scalability, as well as the rise of ROI and reduction of system complexity. Besides, the identified issues in the survey were application decomposition, database partitioning and migration, inter-service communication design and so on. These results were also similar to the previous study. Moreover, they proposed a migration process framework based on real research participants' projects, but a specific technical discussion about the framework was missing in their research.

Moreover, Taibi et al. [98] further proposed an assessment framework to support companies to make decisions and evaluate the migration to microservices. This framework included several steps for metrics identification, migration decision-making, and actual migration process. The main metrics were identified from an industrial survey and referred to ISO/IEC 25010 standard.

Lastly, Carvalho et al. [17] conducted a survey with specialists to identify the criteria which they thought essential during migration. They stated that the most critical criteria included coupling, cohesion, reuse potential, and requirements impact. Some other criteria such as database schema were also influential to the migration process. What's more, they pointed out the need for more formal techniques and tool supports for migration.

Based on the above studies, several gaps were identified. First, a structured overview of the supporting techniques and their benefits as well as limitations was missing. Second, a microservice migration framework focusing on technical methodologies was also lacking. Third, considering the fact that microservice is gaining growing attention these years in the industry, previous studies that were conducted mostly around 2017 or earlier may be obsolete. Therefore, a new meta-study within this area was desired. These were the purpose of this master thesis.

Chapter 3

Systematic Literature Review

In order to continue the study based on the acquired knowledge, the current state of research about architectural refactoring to microservice should be reviewed and documented. Therefore, in this thesis, I defined the goals to be identifying the newly released contributions which described refactoring approaches and had not been captured by previous meta-studies [33][34][83] yet. These approaches gathered from the contributions should be analyzed using a unified framework similar to the one used in [33]. Furthermore, this framework would be adapted according to the new findings of my research. Particularly, emphasis should be put on the techniques used for decomposition into microservices.

3.1 Research Methodology

In this thesis, I will introduce a formal systematic literature review process for software engineering. This methodology referred to Fritzsch, J. and his colleges' report [33] and also a generic guideline proposed by B. Kitchenham and S. Charters [52]. The purpose of this methodology is to follow a clear and concrete process, summarize the empirical evidence related to one technology, identify gaps in current research, and also enable researchers to continue the research more easily [52]. Due to the well-defined methodology, it requires considerably much more effort than normal research. However, such endeavor is worthwhile because all relevant empirical evidence will be collected by means of meta-analytic techniques [52] and presented as a comprehensive overview. Additionally, it can identify research gaps of the current study and refine researcher's knowledge on the target area [81]. What's more, the acquired results are reproducible, more consistent and less likely to be biased because various results including positive and negative ones can be analyzed by statistical techniques after contribution reading.

3.2 Research Protocol Definition

The first step of the systematic review was defining a research protocol. Considering the goal of our meta-study, the review should be able to find a sufficient amount of contributions that were related to our topic, that was, about microservice migration.

In order to answer the research questions that I have proposed in Introduction chapter, the research methods were applied accordingly. In this thesis, I first defined the target databases: ACM Digital Library, IEEE Xplore, Springer Link, and Google Scholar. ACM Digital Library is a platform for research and networking, which contains an extensive bibliographic database focused specifically on the field of computing [60]. IEEE Xplore is also a worldwide database dedicated to advancing technology, including information technology and software engineering, offering highly cited contributions [43]. In addition, Springer Link is also a technical and scientific portfolio containing various publications covering the area of computer science and software engineering [96].

These databases were selected because they are famous and common databases accessible by scholars in the field of this topic. They provide relatively high-quality full-text contributions and also they cover nearly all relevant knowledge, which offer a comprehensive scope about my topic. In combination with these three databases, many kinds of contributions such as conference proceedings and technical reports were scanned. Therefore, unbiased results were achieved because both supporting contributions about effective migration approaches and opposite ones that documented immature or failed attempts were involved. What's more, Google Scholar was also chosen as a supplementary source of knowledge, considering the fact that there might exist some contributions of interest, but they were unavailable from the above three databases. Other database sources such as Elsevier were also considered to be included. However, due to the fact of workload and schedule, I excluded them later during the research. I believed that the current search scope could already provide me an adequate amount of contributions for my study. Therefore, I suggest that other databases should be included, and then more extensible results can be collected for further research.

All information could be later gathered and analyzed as the knowledge base, and then it would support web-based tool development and further serve the software architects, developers, and researchers.

For the second step, the research keywords were defined as representatives of my study scope. In Table 3.1, this case of study mainly focused on "microservice", and the contribution under investigation should be about architectural refactoring, so I included several possible empirical practices related to it, for instance, "migration", "evaluation" and "adoption". It would be a plus if the contributions had documented sections about the comparison between these two software structures in case of the development process, performance, tool support, etc., so "monolith" was also chosen as a keyword. Last but not least, the expected outcome of our searching was a collection of frameworks, approaches, and techniques, which could give me extensive empirical evidence and support me to define a comprehensive microservice migration framework in this thesis.

Table 3.1: Search Keywords

Subjects	microservice
Practices	migration; evaluation; adoption
Comparison	monolith
Outcome	framework; approach; technique

By referring to the keywords and previous research by Fritzsch et al. [33], a general search string was decided as the input for databases:

("microservice" OR "micro-service*" OR "micro service*") [AND "monolith*"] [AND ("refactor" OR "transform" OR "migrat*" OR "decompos*" OR "partition*" OR "granular*" OR "evaluat*" OR "compar*" OR "adopt*" OR "metric")]*

With the purpose of avoiding missing relevant contributions during the searching process, I tried to include as many synonyms as necessary, such as "refactoring", "transformation", "partition", etc. Additionally, I included "comparison", "granularity", "metrics" and so on, so that I could also acquire some contributions which investigated detailed aspects like granularity decision, microservice quality metrics, migration intentions, and architectural comparison to legacy systems.

Thirdly, explicit inclusion and exclusion criteria were determined to indicate which kind of contributions should be selected for this research. Considering the scope and objectives of this thesis, I determined the inclusion and exclusion criteria as:

CR 1: The contributions under consideration should be published since 2017 so that the research was based on previous literature reviews [33][83] could be continued.

CR 2: The contributions must have shown empirical evidence (practical report), in other words, they must describe approaches of migration from monolithic application to microservices.

CR 3: The contributions should be written in English.

The following important step was defining a data extraction form. The purpose of this step was to design a unified form so that all relevant information that was valuable to this research could be gathered and documented from each selected contribution. This form was of great importance due to the fact that it would store all empirical evidence, and it would construct a repository for the microservice migration framework and the web-based tool. Hence, considerable effort was spent to design and improve the content of the extraction form so that it reflected the requirement of information, which was of most interest for our research.

Fritzsch's study [33] provided a well-grounded overview of microservice migration. And his classification framework was later referred to and improved to create a new data extraction form. In my study, the form was constructed of three main groups: "Paper Context Data", "Empirical Data", and "Quality Assessment". Specifically, "Paper Context Data" tried to capture information about contributions, such as title, year of contribution, author and organization, publisher, available source and so on. "Empirical Data" collected information about approaches, techniques, tools used for migration, as well as drivers for migration, expecting goal and quality after migration, required inputs, etc. Finally, "Quality Assessment" intended to evaluatd feasibility, comprehensiveness and quality of the contributions. This part showed validation aspects, creativeness of the proposed approach, process formalness, and level of detail. Figure 3.1 shows the content of data extraction form. In this research, the form was created using Microsoft Excel.

The first draft of the data extraction form experienced further improvement after discussion with Mr. Fritzsch. In the end, the sections were arranged and merged again, and some other form elements such as process strategy, decomposition strategy, technique sets for migration were added so that in the future it would be much easier to conduct data collection, and all valuable and informative data could be gathered using this form during the systematic literature review.

Emphasis should be put on some specific rows in the form which contained new elements other than previous researches. To start with, according to Figure 3.1, in the "FRAMEWORK/APPROACH" section, the row "The contribution proposed a framework or approach?" should be clarified. Secondly, the row "Approach" should contain a brief description of general concepts related to microservice migration measures. Especially if the approach was bound to certain programming languages, software frameworks or design patterns, it should be annotated in this row as well. Thirdly, "Process Strategy" was further classified into six main categories: rewrite, extension, strangler pattern, continuous evaluation, splitting the existing code base, other strategy. In this research, process strategy was closely related to coding and development processes. It indicated how should architects and developers utilize application resources and handle practical programming and migration tasks. On the other hand, the row "Decomposition Strategy" focused more on application functional decomposition. In other words, it was more related to architectural decomposition for different functional clusters. Therefore, a set of widely-used decomposition methodologies were documented, including Domain-Driven Design, functional decomposition, using existing system structure and so on. Additionally, "Technique Sets" were recorded to illustrate the practical procedures step by step. This could provide architects and developers a clear understanding of how to migrate monolithic applications in a given context.

Also, "Quality Assessment" was merged into "Empirical Data" group for the sake of simplicity and understandability. The "Validation Type" involved: experiment, example, and case study. After the proposal of an approach, it was favorable to validate it to prove its feasibility and correctness. For instance, experiment and case study are similar because they require to test the approach with practical migration tasks, say, an existing banking system or parking charging system. But experiment may be less formal and the monolithic application may be relatively small, while case study requires much more effort and the microservices will be actually put into service after migration. In contrast, the researchers may provide a hypothetical monolithic application,

DATA EXTRACTION FORM

Data item	Value
PAPER CONTEXT DATA	
Title of Contribution	
Year of Publication	
Authors of Publication	
Publisher	
Avaliable at (ACM, IEEE, ...)	Online Sources
Organization, Company	University/Company/Research Group
FRAMEWORK/ APPROACH	
The contribution proposed a framework or approach?	framework or approach
Approach	1) Approach description 2) Generic or bounded to a certain programming language/framework/pattern?
Process Strategy	(Rewrite, Extension, Strangler Pattern, Continuous Evolution, ...)
Decomposition Strategy	Domain-Driven Design, Functional Decomposition, Existing System's Structure, Other (or non-systematic)
Technique sets for migration (including clustering decision) Can be applied separately?	
Technique Type(s)	SCA/ MDA/ WDA/ DMC With a basic description
Applicability	Applicable senarios and suitable system architecture description
Atomic Unit, Granularity	Minimal unit that can be the result of a decompostion
Input	
Output	
Tool Support	1) Tools for migration and its support 2) Programming languages/frameworks/patterns are covered by the tool
Validation Type	Experiment, Example, Case Study
Validation Details	1) System information (name, purpose, type, size) 2) Resulting number of services 3) Validation Steps
Validation Metrics	(What quality attributes have been measured/investigated during the validation? Quality attributes according to ISO25010)
Drivers/expectations and result influence for the migration?	Intentions for migration, negative and positive results (metrics)
Score	Five-point likert Scale (1~5) from low to high, reflects the quality of contribution

Figure 3.1: Data Extraction Form Structure

explain its functionalities and consisted modules, and use it as an example to explain migration steps. Validation with example may be less convincing since it is a fictitious scenario and some issues or drawbacks may be ignored which can only be captured in real and complex environments. However, it is easier for readers to grasp the basic concept.

Besides, "Validation Details" stored information about the monolithic application, such as name, purpose for usage (functionality) and size. It also recorded the resulting number of microservices after migration and detailed steps intended for validation.

Furthermore, quality attributes had been measured during the validation, these attributes should refer to [3] and they should be shown in "Validation Metrics", important attributes involved: maintainability, performance, reliability, scalability, security, etc. These attributes indicated the final quality and effectiveness of the microservice migration. And quantifiable matrices were preferred for the sake of conviction. Additionally, "Drivers/ expectations and result influence for the migration?" captured the intentions and results for migration. This also referred to the quality attributes because quality improvement, technique bottlenecks, maintenance cost were common causes and goals of microservice migration. Considering this information, we could know whether the final result meets the expectation and goals before migration, and they would also reflect the effectiveness of the approach.

The last row of the form was "Score", this was a five-point likert scale thet represented the quality, comprehensiveness, and feasibility of the proposed approach or framework and if it was worth to be referred. The decision of this score was based on "Quality Assessment" group, which was defined previously in the draft form. By evaluating creativeness, formalness, level of detail, and profoundness of the validation process documented in the contributions, a score was signed by me for reference. It enabled me to distinguish and rank various contributions. Therefore, I found it especially helpful during the systematic literature review. Till now, the data extraction form was completely designed. For detailed explanation about each element in the form, see Section 4.2.

Finally, a formal and precise definition of the research protocol was successfully finished. It played an important role at the initial stage because it could serve as a criterion about what kinds of contributions should be included in the research. More particularly, it acted as the backbone of the microservice migration framework, and it made data extraction, synthesis and quality assessment from different contributions possible. Furthermore, it could be exploited to serve the development of web-based tool and later answer the proposed research questions. I believe that the research protocol plays a fundamental role in almost every systematic literature review, and it may directly influence the result and conclusion of the research. Therefore, considerable effort should be spent during this stage, and issues such as vague definitions of input, different data formats, missing information can be avoided in further research processes.

In the next section, the actual literature review would be conducted. As we can see in Figure 3.2, this proceeded in seven stages: contribution searching, contribution selecting, thesis reading, data extraction, data synthesis, quality assessment, and framework definition. Some steps can be performed in parallel: contribution searching and selecting could be done at the same time; for each contribution, thesis reading, data extraction, and quality assessment were conducted sequentially for each contribution. And after I read all contributions, a general data synthesis process would be done, and a framework would be defined according to all relevant data collected in the form.

3.3 Contribution Search

The purpose of contribution search and selecting was to find and filter contributions that included predefined search terms and answered research questions [52]. Effort was required in order to include as many relevant contributions as possible. Also the publication bias should be minimized. Therefore, I strictly followed the research protocol step by step.

During the contribution search stage, considerable attention must be paid when setting the search strings for different databases. In Section 3.2, a set of research keywords, target databases, and a general search string were defined. However, each target database had a slightly different searching user interface, and they covered different academic fields and contributions, or classified

A Meta-Approach to Guide Architectural Refactoring from Monolithic Applications to Microservices

Examiner: Prof. Dr. Stefan Wagner
Supervisor: Jonas Fritzsch
Student: Qiwen Gu

Project Start: Donnerstag, 30. April 2020
Display Week: 21

	TASK	PROGRESS	DURATION	START	END
PLANNING	Background Study	100%	4	30. Apr.	6. Mai.
	Research Protocol Definition (Research Question, Search Strat	100%	3	7. Mai.	12. Mai.
CONDUCTING	Contribution Searching	100%	4	13. Mai.	19. Mai.
	Contribution Selecting	100%	10	20. Mai.	3. Jun.
	Thesis Reading (Systematic Mapping Study)	100%	20	4. Jun.	2. Jul.
	Contribution Quality Assessment	100%	5	3. Jul.	10. Jul.
	Data extraction (Form)	97%	5	13. Jul.	20. Jul.
	Data Synthesis	100%	4	21. Jul.	27. Jul.
	Framework Defination	100%	4	28. Jul.	3. Aug.
PROGRAMMING	Webdemo Design (Mockup, Class Diagram)	100%	3	4. Aug.	7. Aug.
	Repository Building	100%	2	10. Aug.	12. Aug.
	Web Backend Implementation	100%	4	13. Aug.	19. Aug.
	Web Frontend Implementation	100%	4	20. Aug.	26. Aug.
	Testing	100%	3	27. Aug.	1. Sep.
	Validation	70%	15	2. Sep.	23. Sep.
REPORTING	Master Thesis	0%	20	24. Sep.	22. Okt.
	Master Thesis Refinement	0%	4	23. Okt.	29. Okt.

1 Montag, 14. September 2020
14.09 15.09 16.09 17.09 18.09 19.09 20.09

Figure 3.2: Gantt Chart for Master Thesis

them into different subject categories. So later, when I actually started searching among databases, I found that using the general input string was unfeasible. If the same search string was used in different databases, it was impossible to get the optimal number of results from each database. The reason was that the search string contained too many keywords, and it was too specific for databases to find relevant results.

Hence, I adapted the search string for each database and search for the results iteratively. I aimed at getting the maximum number of matched results. As a result, the search string for each database can be seen in Table 3.2.

Among these matched results of each search, their abstracts were read thoroughly, and they would be filtered based on inclusion and exclusion criteria. For instance, some contributions discussed microservices, but they focused on application development rather than monolith migration; others might be theoretical studies about the microservices, and they showed less practical evidence in the abstract; there also existed some contributions which were written in Italian or French. So these contributions were excluded in the first phase.

3.4 Contribution Reading, Data Extraction and Quality Assessment

The purpose of contribution reading was to read and gather valuable information which was relevant to my topic and classify this information in order to favor data extraction and synthesis.

Therefore, the first step was creating a contribution index. This index contained basic information such as title, the decision of inclusion or exclusion, database source, and note about exclusion reasons. It was essential to the review because it provided an overview of the studied contributions. It also made the research process traceable and clear to researchers. In the future, they could validate the decisions again based on the contribution index as well. The contribution index was available in Appendix A Table A.1.

In this study, I did contribution reading, data extraction, and quality assessment in parallel. After I read one contribution, I would immediately extract the knowledge about proposed approaches or frameworks documented by the authors and filled this knowledge into the corresponding data ex-

Table 3.2: Adapted Search Strings for Each Database

Research Database	**Search string**
IEEE Xplore	[Publication Title:("microservice*" OR "micro-service*" OR "micro service*") [AND "monolith*"]][AND Publication Date:(2017 TO 2020)]
ACM Digital Library	[Publication Title:("microservice*" OR "micro-service*" OR "micro service*") [AND "monolith*"] [AND ("refactor" OR "transform" OR "migrat*" OR "decompos*" OR "partition*" OR "granular*" OR "evaluat*" OR "compar*" OR "adopt*" OR "metric")]] [AND Publication Date:(01/01/2017 TO 05/31/2020)]
Springer Link	[Publication Title:("microservice*" OR "micro-service*" OR "micro service*") [AND "monolith*"]] (Content type is only conference paper, and discipline within Software Engineering or Information Systems Applications (incl. Internet))[AND Publication Date:(2017 TO 2020)]
Google Scholar	[Publication Title:("microservice*" OR "micro-service*" OR "micro service*") [AND "monolith*"] [AND ("refactor" OR "transform" OR "migrat*" OR "decompos*" OR "partition*" OR "granular*" OR "evaluat*" OR "compar*" OR "adopt*" OR "metric")]][AND Publication Date:(2017 TO 2020)]

traction form. Particular emphasis was put on the techniques used for microservice decomposition. Meanwhile, the "Score" was also assigned in the form as quality assessment.

3.5 Contribution Selection

Based on the full-text reading of all contributions, the second filtering was done to refine and narrow the scope of the study. This step was performed by two researchers: Jonas Fritzch and I.

To begin with, I referred to the inclusion and exclusion criteria again to filter out irrelevant contributions such as literature reviews or conceptual solution proposals without detailed migration steps. Next, I created a contribution index that contained information about all included contributions. And this index was checked by Fritzsch again. He further read the contributions and provided his advice about inclusion or exclusion. In that case, the bias caused by one researcher can be minimized.

3.6 Data Synthesis and Framework Definition

The last step of the systematic literature review was data synthesis. As described by Kitchenham and Charters: "The extracted data in the forms should be in a consistent format. And the form should be able to highlight similarities and differences between study outcomes" [52]. Therefore, data synthesis is an essential step to inspect the data and reconstruct them consistently. Thanks to the well-defined data extraction form, little effort for this step was required in this study. Basically, the terminologies used among different contributions and the format such as spacing and punctuation for the content were unified.

After data synthesis, a new microservice migration framework was designed according to the knowledge acquired from the literature review. The proposed framework should provide software architects and developers an up-to-date overview of microservice migration approaches or frameworks that were possibly feasible according to different scenarios. The repository of data extraction forms as well as the migration framework were the final outcome of the systemic literature review, and they would support the development of a web-based tool later.

3.7 Gantt Chart

In an attempt to always keep track of the review progress and manage it in proper order, a Gantt Chart was created using Microsoft Excel, which reflected the whole schedule of the thesis proceedings. According to Figure 3.2, the detailed planning for each stage was listed and organized. Specifically, each stage was described as task name, progress rate, duration, start date, and end date. Besides, all necessary stages which should be performed in the thesis were also listed in the Gantt Chart, including background study, research protocol definition, contribution search, contribution selection, thesis reading, data extraction, data synthesis, quality assessment, framework definition, web-demo design, repository building, web-demo back-end implementation, web-demo front-end implementation, testing, validation, and master thesis writing and refinement.

Till now, the systematic literature review and its methodologies were completely introduced. In the next chapter, I discuss the results of the review and provide some evidence or examples accordingly.

Chapter 4

Results and Analysis of Systematic Literature Review

In this chapter, the result of the literature review and acquired knowledge will be introduced in detail. Some terminologies and concepts were already discussed in Chapter 2, but they will be further extended with new evidence or examples identified during the contribution reading. Also, the first three research questions proposed in Introduction will be answered.

4.1 Result of Contribution Searching

Consequently, the total number of search results and selected contributions after abstract reading are listed in Table 4.1. As we can see, I got a vast number of results from each database, especially Springer Link and Google Scholar, because the searching scopes in terms of academic fields were more comprehensive than other databases. However, after abstract reading, more than half of the contributions were filtered, resulting in sixty-one contributions remaining for my study.

Please note that even though a significant number of irrelevant contributions were excluded in the first phase, some contributions still existed, which failed to meet the inclusion criteria. However, it was difficult to figure them out only by reading their abstracts. Several studies mentioned monolith migration in their abstract or paragraph, but maybe they provided less evidence about empirical practices or their approaches were non-systematic, or they were rather vague and fictional. For that reason, the following stage was contribution reading. All included contributions were full-text read, and final decisions were made whether they should be included or not.

Table 4.1: Total Number of Search Results and Selected Contributions

Research Database	**Total Results**	**After Abstract Reading**	**After Full-Text Reading**
IEEE Xplore	26	17	10
ACM Digital Library	61	14	2
Springer Link	204	17	11
Google Scholar	517	11	6
Snowballing	2	2	2
Total	810	61	31

4.2 Analysis of Contributions

As mentioned earlier, contribution reading, data extraction, and quality assessment were conducted in parallel to improve research efficiency and reduce repeating efforts. In the following paragraph, I will sequentially give explanations and examples of some essential parameters in the data extraction form. Furthermore, I will also introduce some approaches or frameworks identified during the study. Therefore, the first sub-question RQ 1.1 can be answered based on knowledge acquired during contribution reading.

> *RQ 1.1: How can the proposed approaches or frameworks be classified? What strategies and techniques did they implement?*

4.2.1 Required Inputs

The first part to be discussed is required inputs. Currently, microservice migration is still an area with research potential, and most researchers propose their unique views and creative approaches based on their project experience. Considering this fact, available resources of the existing monolithic application play important roles in the decision-making during migration. So it is the starting point of the microservice migration procedure. Typical input resources are source code, use case diagram, system specification, application programming interface (API) and others. These are common resources which most software projects will create during the design and development stage. During my study, I notice that some approaches use single input resources while others use multiple inputs as a combination, and I will introduce them one by one.

(1) Source Code
My study revealed that sixteen contributions chose source code as the main input in combination with databases, system specifications, APIs, etc. They accounted for relatively half of the final included contributions, which meant that this was essential for migration, especially for monolith-migration (brownfield development).

Sarita and Sebastian [91] proposed an approach using Docker [27]. This platform is based on container virtualization engine. It is designed for agile development, deploying, and migration to cloud and microservices. Docker provides three main components: Docker Image as a template for creating Docker Containers; Docker Registry which stores and manages Docker Images; and Docker Container as an independent virtualized environment created from an image. For microservice migration, Sarita and Sebastian divided the monolithic application by separating the front-end and back-end of MVC pattern. And they further divided these modules based on factors such as importance and frequency of change, resource requirements (memory, computational intensity), inter-dependencies (asynchronous messages) and so on. After division, they decoupled the resulting modules by adding REST APIs as interacting interfaces and refactored them into microservices. Next, they built Docker Image for these services and deployed them on containers as microservice instances. Therefore, the instances composed a micro-service application. To use microservices, clients could interact with Daemon process running on the host utilizing CLI commands. Daemon accounts for building, running, and distribute containers. To sum up, Docker makes microservices independent of platform, easy to scale and manage, and enables automatic development and delivery. Sarita and Sebastian argued that it was a good fit for microservice migration [91]. Most importantly, source code was an important resource in their approach for application decomposition and refactoring.

(2) Use Case Diagram
Among all contributions, only three of them used use case diagrams as one of their inputs for structural analysis. Gemino and Parker [35] said that use case diagrams are "simplified and graphical representation of systems". They serve as the "blueprint" of the system and illustrate users' interaction with the system and how users are involved in different use cases.

Santos et al. [90] proposed an approach using use case diagrams for domain modeling, and they also adapted the Four Step Rule Set (4SRS) method [63] during migration. Each use case indicated one or more microservice components. Briefly, 4SRS method was based on MVC pattern. It decomposed the functional requirements of use cases and generated corresponding functionalities of microservices [63]. The first step was requirements modeling in use case diagrams. The functionalities were decomposed based on specific tasks in a tree-like form. They were arranged hierarchically, from high-level business functionalities to low-level create, read, update and delete operations. Second, for each use case, components such as the system's interface, data model, and logic/control were created. Third, redundant components which had the same purpose and functionalities should be eliminated, and developers should add name and description for each remaining component. Fourth, the remaining components were grouped into packages based on business processes they related to, which composed higher-level microservices. Subsequently, developers should define the communications and interactions between microservices by setting up service channels according to specific rules and scenarios. Finally, microservice applications, including service participants, capabilities, architectures, and interfaces, were interpreted using service-oriented architecture modeling language (SoaML). Therefore a clear overview of the whole structure was available to architects and developers.

Even though few approaches exploited use case diagrams during the migration process, Santos et al. [90] claimed that there is a tendency to use languages oriented and use case diagrams to model microservices and operations. And I believe that this is also a straightforward resource for understanding the application's functionalities and can help microservice decomposition comprehensively.

(3) System Specification
System specification, including functional and non-functional requirements, is also a necessary document created during the design process. It defines the functionalities and other aspects such as quality metrics, user satisfaction requirements and so on. This input acts similarly to using case diagram, but the main difference is that it is usually described linguistically.

Seven approaches were proposed by developers that used system specification as input resource. Specifically, one approach introduced by Martin and Boggies [66] reported their experience of conducting microservice migration based on glossary definition in business language. To start with, they defined a glossary of terms to clarify business entities. They set up a shared vocabulary between the customers and the technical team to eliminate any misinterpretation of the design concept during communication. Then they converted the glossary into a domain model diagram, showing entities and their relationships graphically. The following step was to elucidate use cases in the diagram about how users interact with each component to realize functional requirements. Next, robustness analysis was done to improve the descriptions and avoid missing any components. The fifth step was converting use cases into sequence diagrams to inspect specific tasks to be performed by different components of the system and specify interface functions for each use case as the entry points. In order to achieve balanced participation of entities, they applied Aggregated Class Interaction Diagram (ACID) as a visualization tool. Therefore, they could graphically partition boundaries between components and measure coupling, cohesion, and work balance between microservices. They clustered components based on the number of relationships, component size, the variance of amount of functions and so on.

Considering the suitability of clustering decision and granularity, Martin and Boggies [66] admitted that this approach might be relatively crud. However, this was also an inspiring way for migration because it focused on linguistic definitions rather than actual codes, which offered us a higher understanding of the monolithic application. Also, it benefited further functional decomposition and made the clustered entities more consistent concerning business capabilities.

(4) Application Programming Interface (API)

According to the microservice architecture design standard, interactions between different microservices are usually realized by Restful APIs. Therefore, original APIs in monolithic application can serve as a reference for application partition and later be converted into Restful APIs. Surprisingly, only one document mentioned an approach that used API specification as the input for migration.

AI-Debagy and Martinek [5] have decomposed a monolithic application by analyzing its OpenAPI specification. Several algorithms were applied during migration. In detail, OpenAPI specification was imported as an input document, and the operation names was extracted. Next, the names were converted into word vectors. To achieve this, FastText or Word2Vec models were trained and utilized. These were AI models trained with different data sets. The average of word vectors in every operation name was calculated, which meant to sum up each word vector and then divided itself by the number of words in each name. This mid-term result was fed to the Affinity Propagation Algorithm [32] to find the number of microservices by measuring messages between data points. Then similar data points and exemplars (indicating a possible microservice point) were grouped. Therefore, candidates of microservices were generated for developers to aid the decomposition of the monolithic application. Finally, data consistency within clusters and performance of decomposition method were validated by measuring metrics such as Silhouette coefficient [89], precision, recall and F-measure metrics. What's more, AI-Debagy and Martinek implemented the method using Python in combination with text analysis and clustering libraries. They conducted experiments to migrate Amazon Web Services, PayPal, Kanban Board and Money transfer app. After comparing their result with the actual number of microservices of each application, they proved that the method could produce relatively correct decomposition and better performance since their method was nearly automatic.

Microservice migration based on APIs has advantages such as better performance, consistent results and comprehensive applicability because API specification is usually platform-independent. Developers themselves can also decide the implementation of algorithm. As long as the API specification is available, this resource and approach can be considered a proper option. It is worth further research to make the algorithm more mature with regard to word vector conversion and clustering [5].

(5) Other Inputs
Besides the input types I have discussed already, other resources are used in some specific scenarios. For instance, Alwis et al.[25] used source code and database to figure out the run-time relationship of system components; Jin et al. [47] exploited monolithic application with test cases, executed them to inspect the log file and clustered the application by functionality oriented microservice extraction (FoME) method; More than three contributions [87][20][57] claimed that they selected source code, system requirements, design documents, and some additional information such as developers' interview or log files for migration; Others [64][42][109] might use conceptual design diagrams like service contract diagrams, structural diagrams, or UML diagrams to graphically decompose the monolithic application.

Generally speaking, the selection of input sources is an open answer. So it is suggested that architects and developers can include their available sources and explore potentials among them to benefit the migration process, improve quality and reduce effort. Because there are vast input resources for different software projects, there is a growing need for a general migration framework that can enable architects and developers to quickly find out possible migration approaches. Again this is the purpose of my study to design such a framework and fill this gap.

4.2.2 Expected Output

After the execution of the approach, it is also interesting for architects and developers to know what kinds of possible outputs will be generated in the future. Ideally, a fully functional microservice application is desired after migration, so automatic migration approaches and techniques are

Application	Proposed Decomposition	Optimal Decomposition
Money Transfer	addToAccountUsing createAccountUsing getAccountUsing	addToAccountUsing createAccountUsing getAccountUsing
	createCustomerUsing getAccountsForCustomerUsing getCurrentUserUsing getCustomerUsing getCustomersByEmailUsing getTransactionHistoryUsing	createCustomerUsing getAccountsForCustomerUsing getCurrentUserUsing getCustomerUsing getCustomersByEmailUsing
	doAuthUsing	doAuthUsing
	moneyTransferUsing	moneyTransferUsing getTransactionHistoryUsing
Kanban Board	doAuthUsing	doAuthUsing
	getBoardUsing listAllBoardsUsing saveBoardUsing	getBoardUsing listAllBoardsUsing saveBoardUsing
	backlogTaskUsing completeTaskUsing deleteTaskUsing listAllTasksUsing saveTaskUsing scheduleTaskUsing startTaskUsing updateTaskUsing	completeTaskUsing deleteTaskUsing listAllTasksUsing saveTaskUsing scheduleTaskUsing startTaskUsing updateTaskUsing
	getHistoryUsing	backlogTaskUsing getHistoryUsing

Figure 4.1: Candidate List of the Proposed Decomposition and the Optimal Decomposition [5]

favorable. However, my study reveals that such kind of approaches remains rare. So in this study, the types of outcomes generated by each approach were documented and classified. In the future, when the users refer to my migration framework, they can evaluate whether the approach under consideration can meet their expected goal. Also, they can try to minimize gap between actual outcomes and expected ones. My study revealed that six contributions generated candidate list of microservices; while eleven constitutions suggested detailed microservice structure. The remaining contributions provided other types of outputs as results.

(1) Candidate List of Microservices
The Candidate list indicates the possible decomposed microservices after migration. It acts as a "check list" for architects and developers to assist their actual partitioning of the application. The partitioned clusters include data entities, methods, interfaces, and other system components. Within every cluster, the entities are either functional-related or highly cohesive with each other.

Previously, AI-Debagy and Martinek [5] have introduced their migration approach based on OpenAPI specification. As we can see in Figure 4.1[5], the algorithm was able to generate a clear list of decomposed components for each monolithic application during their validation process. Different APIs were grouped according to name similarities. The size of them varied and was decided by the algorithm automatically. Additionally, they compared the proposed result (left) to the real-life decomposition example (right).

However, despite its clear structure, one drawback of the candidate list is that the relationships and interactions between each microservice are absent. This may bring inconvenience during microservice decomposition optimization because development teams may find it challenging to check the coupling and communication behavior between components.

(2) Microservice Architecture
Micro-service architecture as output solves the problem of missing relationship between each component, since it illustrates not only microservice components in the application but also

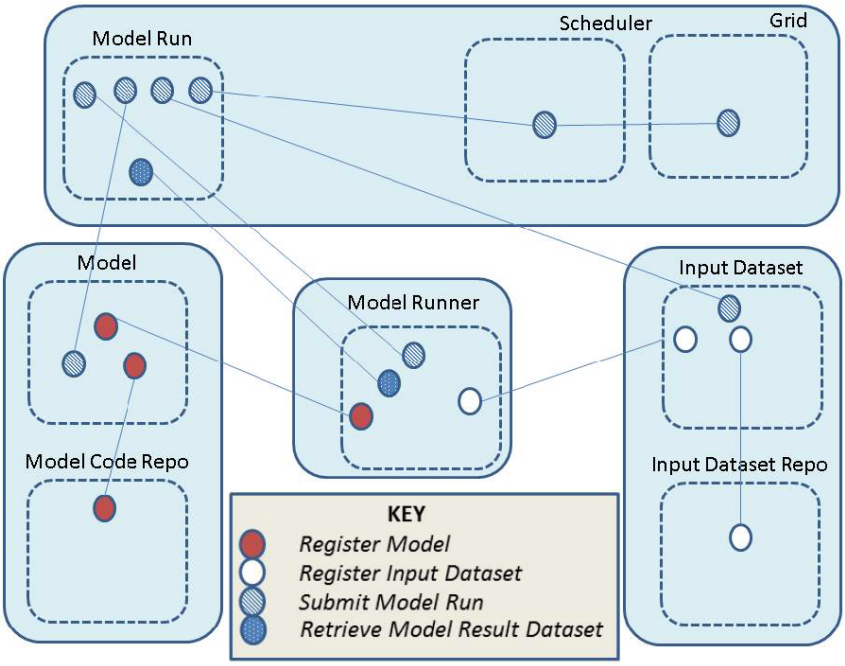


Figure 4.2: Decomposed Microservice Architecture [66]

interactions between them. Therefore, a comprehensive architecture design is available to architects and developers to aid them during the migration process.

In "Required Inputs - System Specification", the approach proposed by Martin and Boggies [66] has been discussed. In order to partition the application properly, a visualization tool (ACID) was used. In the end, a graphic structure of the microservice application example was generated by the tool, which can be seen in Figure 4.2[66]. The components of application were clustered into four microservices. Furthermore, each of them had specific business entities identified from the monolithic application by system specification analysis and use case analysis. According to their application, "register model" represented code repository, "register input dataset" indicated the input data repository, "submit model run" meant to execute the code with input, and "retrieve model result dataset" showed the link to result and its status after execution. In the figure, the interactions between business entities of the microservices were clearly presented by lines. Using this architecture, such interactions could be optimized by adjusting the partition and observing the result again. And this could be done with help of the tool or manually.

In brief, the microservice architecture, which is a graphical representation of the migrated application, is usually much straightforward than the candidate list. This may explain why more development teams have chosen microservice architecture as the output after migration.

(3) Other Outputs
Other than the previous two outputs, some approaches produced particular files or documents after migration. Two approaches [51][91] applied virtual machine technologies such as ENTICE or Docker for microservice deployment, and final outputs were images for containers that were runnable on virtual machines.

Additionally, there were four proposed approaches [87][108][100][42] which generated partition diagrams of services as results. The main difference between this diagram and microservice architecture is that: the microservice architecture shows the decomposition of system com-

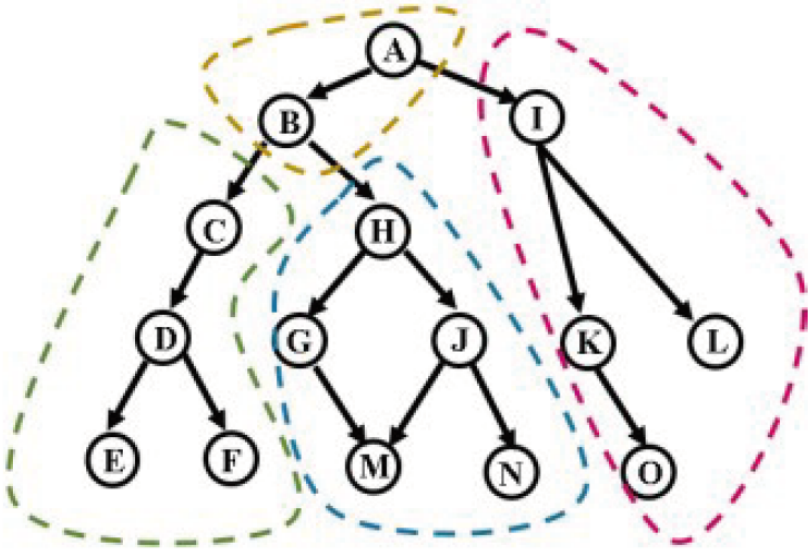


Figure 4.3: Service Partition Diagram [87]

ponents, while the service partition diagram focuses more on the decomposition of service (task) execution processes. Take Figure 4.3[87] as an example, the partition diagram groups execution processes if they are with in the same or similar service (task) provision flow. The developers can further partition interfaces, services, and databases according to this diagram and deploy microservice application.

The remaining approaches might give suggestions about final microservice application design, which were decided by the algorithm, but they were rather vague about the detailed outcomes. In conclusion, the desired outputs also vary according to different scenarios. So the choice of this section is also left to be free to development teams based on their needs.

4.2.3 Technique Type

As previously mentioned, Fritzsch [33] has grouped technique types groups into four main categories: Static Code Analysis Aided (SCA) method, Meta-Data Aided (MDA) method, Workload-Data Aided (WDA) method and Dynamic Microservice Composition (DMC) method. Each technique type has specific procedures. Based on the given input resources, various analyses can be done in order to construct the structure and behavior of the monolithic application. This provides a guideline about how to partition it. Usually, one or more technique types were applied in one approach to conduct a comprehensive analysis of monolithic applications.

(1) Static Code Analysis Aided (SCA)
Static Code Analysis Aided (SCA) method analyzes and decomposes applications by inspecting their source code, so that the static traits of the application can be understood, including business objects, business functionalities, context boundaries, relationship between different classes, access to databases and so on.

Pigazzini and her colleagues [82] analyzed the Java code bases statically. They used a tool called Arcan, which was developed for the migration process. First, the tool conducted architectural smell detection to find inappropriate dependencies such as cyclic dependency, multiple feature concentration in one component, etc. This could filter out architectural smells and microservice candidates were reserved. Second, they inspected the dependency

graph to find out structurally and functional independent class groups that could be transformed into microservices. In addition to code structure analysis, they also applied machine learning algorithms like Latent Dirichlet Allocation(LDA) [11] and Seeded Latent Dirichlet Allocation (SLDA) [45] to extract domains of the project from comments and source code words in Java classes. Finally, they got hints about microservice grouping.

Static Code Analysis Aided (SCA) method is a commonly adopted approach, twenty-two contributions have performed static code analysis of monolithic applications. This is because source code is one of the most informative and straightforward resources that show monolithic applications' behavior and structure. In the future, it is also an important and solid resource to construct the microservice application.

(2) Meta-Data Aided (MDA)
Meta-Data Aided (MDA) method inspects the application from conceptual resources, such as UML diagrams, use case descriptions, system specification, etc.

Sayara et al. [92] focused on a generalized approach that defined business capabilities through multidimensional update rate and scaling rate of the monolithic system. According to their suggestion: "A service should be assigned with one specific responsibility. The homogeneity of each service is decided with respect to the update rate, scaling rate and technology used" [92]. So they broke down all business capabilities and calculated metrics from the update rate and scaling rate of each service. The update rate indicates the frequency of update. Likewise, the scaling rate shows the possibility of scaling up or down of a service. These aspects were chosen as decisive factors because they reflected the essential characteristics of microservice architecture: scalability, maintainability, high cohesion and loose coupling. Next, an algorithm was provided in order to calculate similarity between capabilities regarding previous metrics. Finally, Multidimensional Scaling Technique (MDS) was used to group the sub-business capabilities having similar weight. This indicated that the grouped parts showed similar business functionalities, and they used same technologies. As a result, a microservice structure diagram of partitioned service and databases was generated.

I found that six contributions have [92][38][22] applied this technique. The number was relatively small because resources like UML diagrams and system specifications were sometimes unavailable. Also, these documents alone might be difficult to construct a comprehensive view of monolithic applications. Therefore, development teams usually apply Meta-Data Aided (MDA) method together with other techniques to inspect the system.

(3) Workload-Data Aided (WDA)
Workload-Data Aided (WDA) method observes applications' operational data, including log files about method invocation, data access and so on. Analyzing these data can get information about the application's run-time behavior, performance, and communication intensity among different components. This information can also aid further migration and solve existing problems such as performance bottleneck and finding suitable service boundaries.

As mentioned in "Required Inputs - Other Inputs", Jin et al. [47] migrated their monolithic application by executing the system with predefined test cases. They also inspected the log file using Kieker, a tool for execution monitoring. The log file contained attributes describing each execution record. Five of them were essential for execution trace generation: "Method" denoted the invoked method; "SessionID" and "TraceID" were unique identifier labeling a session and a execution trace, they indicated specific execution of functions within a session. "Eoi" and "Ess" were the order and depth of the calling stack of methods. Next, the log files were analyzed to create function-level traces and class-level execution traces. These traces indicated the call relationship and order between functions or classes. They were of different granularity, and they provided evidence for further microservice clustering. This will be discussed later in "Process Strategy - Splitting the Existing Code Base".

Workload-Data Aided (WDA) method observes monolithic application's behavior dynamically. In my scope of the study, ten contributions have applied this technique during migra-

tion. In addition, many approaches applied Workload-Data Aided (WDA) method together with Static Code Analysis Aided (SCA) method in order to inspect both static and dynamic characteristics of an application. So they could have a more comprehensive view of the application and make clustering more suitable to their requirement.

(4) Dynamic Microservice Composition (DMC)
The principle of Dynamic Microservice Composition (DMC) method is similar to Workload-Data Aided (WDA) one. Additionally, it improves service partitioning iteratively and dynamically by observing run-time workload and environment.

Nakazawa and his team [74] used a visualization tool for designing microservice applications from monolithic applications. They referred to profile data, source code, and commit information in the project for analysis. Specifically, a calling context tree (CCT) was created using profile data. It indicated the number of communications between microservices. Also, an algorithm was used to compact CCT according to function names so that unnecessary calling contexts such as software libraries were filtered out. Nakazawa explained that "these contexts could be deployed without REST API calls by other microservices, so they were out of consideration" [74]. After that, two initial designs of microservices were generated by employing semantic clustering and CCT-based clustering. The first technique used source code text for class-level clustering, while the second one considered number of communications between functions and performed function-level clustering. These were done by the visualization tool automatically, and it visualized the designs as dependency graphs. Then, it required users to manually refine the design by creating, moving, cloning and keeping microservice candidates. The refinement actions were suggested by the tool whenever the user made changes. It aimed to help the user best reduce the number of API calls within the applications and therefore make the whole architecture loosely coupled. In the end, a visualized microservice architecture design was available to users to help them migrate the monolithic application.

Nearly seven contributions adopted Dynamic Microservice Composition (DMC) method, Nakazawa claimed that it helps developers to gradually improve the architecture and finally meet the requirement in the greatest possible manner [74]. The feedback during whole process is timely and intuitive to the development teams. However, She also pointed out one drawback that their technique might be time-consuming. Additionally, some researchers developed approaches supported by more automatic algorithms and tools to accelerate the optimization process and save time. Take the contribution by Alwis et al. [25] as an example, they applied machine learning algorithms such as Non-dominated Sorting Genetic Algorithm II (NSGA II) and SYNPOP function to achieving global optima. Briefly speaking, this is also one of the common technique types applicable during migration. It has the potential to be further investigated and create various algorithms and tools to make migration design more accurate and fasten in process.

Besides, only one contribution used other technique types [68] and it was vaguely documented in the paragraph. The researchers deployed the application using Docker without specifying detailed steps. In conclusion, my study reveals that nearly all contributions' technique types can be classified into four main categories (SCA, MDA, WDA, and DMC). Moreover, software developers can exploit any available input resources and combine multiple techniques during migration to construct a detailed overview of the monolithic application. They can analyze its static structure and runtime behavior and apply the acquired knowledge to generate the microservice architectural design. Then they will transform the application into a modernized microservice one using decomposition strategy and process strategy.

4.2.4 Decomposition Strategy

Decomposition strategy sounds similar to process strategy, but it represents a higher level of abstraction, that is, it focuses on handling and decomposing application functionalities rather

than practical programming. Nowadays, most applications consist of different components such as classes, modules, libraries, and packages. They interact and collaborate with each other to perform specific tasks. In this case, the application can have many advantages such as clearer structure, less coupling, easier code sharing, better maintainability and so on. Decomposition strategy exploits application partitioning by splitting the application into several components. Each of them represents an individual microservice. In my study, three principal decomposition methodologies were documented:

(1) Domain-Driven Design (DDD)
The approach documented by Fan and Ma [20] applied a typical Domain-Driven Design method during migration. They analyzed the internal system architecture according to source code and system requirements, and they performed Domain-Driven Design method to define bounded contexts and extract possible candidates of microservices. Then they analyzed the system again using database schema to identify foreign keys as possible microservice candidates. These two results were compared, and inconsistent candidates were filtered out. Next, the source code related to the remaining candidates was extracted. After that, the corresponding communication protocol, data format, and microservice architecture were designed. In addition, the database structure was also adjusted to fit the architecture, and the interfaces between services were transformed into Restful APIs or MQTT. The final step was the actual development of microservices accordingly.

Domain-Driven Design is a widely used strategy in monolithic decomposition because nearly one-third of final included contributions have introduced it. As mentioned before, it has the technical benefits, for instance, clearly defined boundaries between business contexts and corresponding functionalities, good maintainability, and it can guide the decomposition of the monolithic application.

However, Microsoft [70] states that it also has disadvantages: in order to maintain the identified domain models, isolation and encapsulation of them must be implemented thoroughly, resulting in a relatively high cost for development. It is recommended if a project's domain is complex and large in terms of system size and team size.

(2) Functional Decomposition
Functional decomposition is similar to Domain-Driven Design method. Both of them focus on decomposition based on systems or business functionalities, but functional decomposition may be less complex in the process. Besides, rather than involving domain experts, development teams alone is enough for designing new microservice architecture.

In "Required Inputs - Application Programming Interface (API)" and "Expected Outputs - Candidate List of Microservices", AI-Debagy and Martinek [5] have decomposed a monolithic application by analyzing its OpenAPI specification using FastText or Word2Vec word vector converting methods, Affinity Propagation Algorithm [32], and Silhouette coefficient [89]. As a result, a complete candidate list of microservices was achieved after decomposition. I classified their approach as functional decomposition because they partitioned different APIs based on name similarities. These names reflected the actual functionalities provided by the application.

Within all final included contributions (thirty-one contributions), twelve of them applied functional decomposition. This is because this strategy produces consistent results with the requirements of microservices, such as low coupling, high cohesion, clear separation of business responsibilities, and explicit light-weight communications. Its procedure is less strict than Domain-driven Design, which applies to many relatively small and simple software projects.

(3) Using Existing System Structure
Instead of decomposing the application and designing a new architecture for it, some approaches adapted a much easier way, that is, using the existing system structure and simply transferring its components into microservices.

According to my statistics, three contributions have used this strategy. For example, Kamimura et al. [49] examined source code and application data of their monolithic application. The program names or class names were defined as entry points, and programs related to each entry point were defined as program groups. Then they extracted dependencies between program groups and dependencies between program groups and data. What's more, the run-time information of program calls or data access were collected. Next, they applied SArF software clustering algorithm [55] with data access [107] to cluster program groups and corresponding data. This algorithm produced a dendrogram as well as an abstract tree model of features accordingly. Subsequently, they arranged the clustered classes and features into a City Block Diagram and assigned a color to each class to indicate its original belonged package. After this, the application data were analyzed by the dedication score to determine the dependencies on program groups. Therefore, the data could be grouped into related program groups. In the end, a visualized map of candidates of microservices was generated by SArF Map. Moreover, the practical migration could be done by partitioning the code and data according to this map.

Because this strategy is highly related to the original structure in class-level or package-level, I define it as "Using Existing System Structure". The decomposition of monolithic application may be course-grained, and the final structure of microservices might be similar to the original one. So the development teams may spend less effort and time to get familiar with the new architecture. However, the quality of the microservice application depends on the original quality of the monolithic one, especially the rationality of the class structural design, such as methods, data access, coupling and cohesiveness among them. Bad performance of the monolithic application will be reflected or even amplified in the final microservice application. Therefore, adjustment and continuous improvement of design is necessary to achieve a good result.

(4) Other Decomposition Strategies
Other than the above strategies, Bucchiarone et al. [13] applied Model-Driven Design strategy to partition their legacy application using JetBrains MPS, which was a text-based meta-programming system. It processed source code and provided projectional editors for developers to generate microservices almost automatically. And its outcome was Jolie-based microservices (Jolie was a programming language for defining microservices by JetBrains MPS). Bucchiarone and his colleagues first imported Java source code into MPS editor and parsed it into MPS Baselanguage. Second, they implemented Microservices Miner to search for microservice candidates in the abstract syntax tree (AST) of Java models. Third, Microservices DSL (domain-specific language) was used to create models for the identified candidates. Forth, Microservices Generator, which included three parts: Microservice Text Generator, Interface Text Generator, and Docker Tex Generator, was exploited to generate corresponding files for developing and deploying microservices in Docker Containers. As a result, microservice architecture was successfully created and ready for deployment in Docker Container.

In addition, five approaches [74][100][65][64][93] mentioned that they conducted class-level clustering or function-level clustering according to functional similarities, data relationships, communication intensities, function call graph, and so on. Barros et al. [13] analyzed and clustered business objects in combination with data access history, execution logs, and call graphs during the migration process. My study revealed that the development teams tended to use formal processes and define models to address problem areas during application decomposition. Such approaches had the benefits of clear boundaries among microservices candidates. Also, some design tools such as JetBrains and SArF Map were available to speed up the efficiency and improve final quality. In short, a proper and well-defined decomposition strategy can set a solid foundation for further implementation stage. It assures that the final result can be consistent as required, and quality aspects after migration are possibly guaranteed.

4.2.5 Process Strategy

As mentioned earlier, process strategy represents the actual utilization of application resources and is closely related to practical programming during migration processes. Once the decomposition strategy was decided and the primary design of microservice architecture was conducted, one or more process strategies can be applied to realize the architecture. In my study, I classified them into six categories, and they will be enumerated sequentially.

(1) Rewrite
Rewrite seems to be an easy way of migration. Indeed, it is highly compatible with almost any approaches, as long as a primary microservice architecture or candidate list is available after decomposition. It can get rid of the existing problems and make a "fresh start" again. Pawlaczyk [78] argued that this strategy is usually considered if developers find: the legacy code is too cumbersome to maintain and too difficult to understand; debugging and bug fixing become burdensome; changing the code or adding new functionalities may cause various errors; new technologies and languages can solve the existing technical and performance problem much easily; a new architecture is adapted for the application...Therefore, Heusser [41] claimed that some programmers tend to choose code rewrite, hoping to solve these problems straightforwardly.

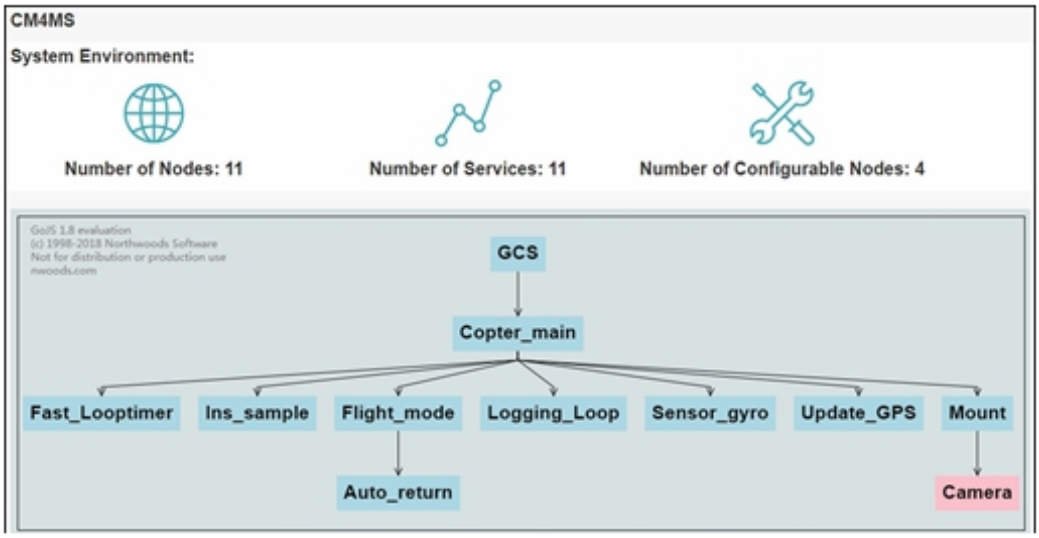
Three approaches have adapted rewrite as their process strategy. Gouigoux and Tamzalit [38] reported a microservice migration project conducted by a software vendor MGDIS SA. They decided to rewrite the legacy application using a modern web-based architecture in 2013, and after three years, the new microservice application was stable and open to all users. Business capabilities were analyzed and decomposed to generate candidate microservices. Additionally, granularity was decided according to the cost of quality assurance and deployment, which were measured by time spent for validation and deployment. These services were deployed using Docker, and they were integrated using Webhooks. Webhooks provided light-weight and HTTP-based communication methods between APIs, making low coupling and simple passive choreography of the microservices possible.

Heusser [41] has introduced some advantages of rewrite: exploiting new technologies, platforms, and markets; using familiar languages and tools for development; better performance and maintainability, etc. Nevertheless, greater development effort, higher risk of failure, and splitting resources to maintain the legacy code during the rewrite process are also the main drawbacks. This may explain why only three contributions mentioned this strategy. It seems that rewrite is an unpopular choice, and other strategies can better solve the problem at lower cost.

(2) Extension
Extension strategy focuses more on adding new features or new components to the legacy application during the migration process. So that the new microservice application can achieve better quality, more functionalities, better maintainability, etc.

Sun et al. [97] introduced their principle approach to migrate a UAV flight control IoT system and make it dynamically re-configurable with various functionalities. First, they applied top-down domain analysis to determine service boundaries. Then, a bottom-up analysis was done of source code, metadata, and database. This could refine the determination of boundaries. Third, a hierarchical layered structure of the monolithic application was reserved and masked into the new microservice application. Additionally, real-time embedded modules for operation control and event triggering were also integrated into microservices. And its performance was ensured to realize the same quality requirement as the legacy system.

Basically, the decomposition and migration of this approach were based on the original object-oriented programming source code and structure. Moreover, Figure 4.4 shows a tool called CM4MS which was developed by Sun and others. Its main features included business process representation, microservice management, and dynamic configuration. In detail, decomposed microservices were registered in the Registration Center, and a layer of key-value store kept



Variation Points

index	Tag	Invoke	Submit
VariationPoint1	Initialization timer	Fast_Looptimer	Submit
VariationPoint2	Initialize the inertial navigation system	Ins_sample	Submit
VariationPoint3	Current flight mode	Flight_mode	Submit
VariationPoint4	Print log	Logging_Loop	Submit
VariationPoint5	Gyroscope information	Sensor_gyro	Submit
VariationPoint6	GPS positioning service	Update_GPS	Submit
VariationPoint7	Mount the device	Mount	Submit

Figure 4.4: UAV Microservice Application with Dynamic Configuration [97]

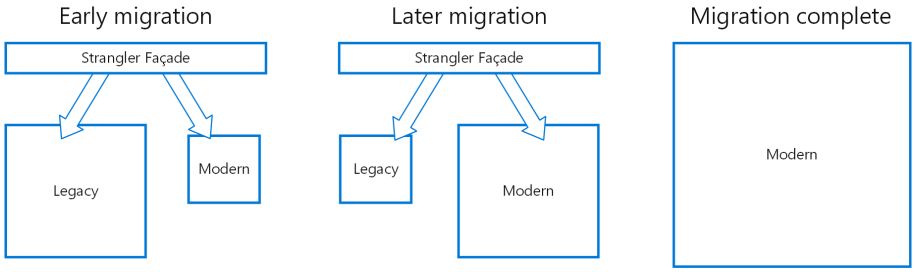


Figure 4.5: Strangler Pattern [14]

invocation relationships between microservices. Therefore, a functional microservice application was composed. In order to achieve dynamic configuration, users used HTTP APIs to update key-value pairs at run-time. As a result, new microservices were introduced, and the functionalities were changed according to user's need. In that case, a re-configurable microservices composition was realized. According to Figure 4.4, the layered architecture of the microservice application is displayed in CM4MS. Each rectangular represents a microservice with specific functionality. The user can change the "Invoke" column in "Variation Points" to switch service components and therefore change the functionalities (mode) during run-time.

In my study, three approaches have applied the extension strategy. If developers want to make the new microservice application flexible, extensible and manageable in the future, they can consider this approach to make the architecture loosely coupled and functional cohesive. Therefore, changes and updates are isolated to other services, the errors caused by changes can be minimized as well.

(3) Strangler Pattern
According to Microsoft's definition [14], strangler pattern requires the developers to gradually replacing part of functionalities with new services, until the new microservice application replaces the legacy one completely. This strategy is similar to rewrite, but it emphasizes incremental replacement, because complete replacement and rewrite are heavy tasks. It is applicable if the legacy application is too complex to maintain or hard to add new functionalities. The old system may be kept running to provide features during migration. Once the new microservice component is developed, the application will locate and point to the new component and discard the old one. To realize this, a façade is created to route user's requests either to new services or legacy ones. The users always use the same interface to interact with the application, and the routing process is transparent to them. The whole migration procedure is shown in Figure 4.5, the size of the "Legacy" part and "Modern" part indicates their percentage in the application during migration. In the end, the "Modern" code will replace the whole legacy part after migration.

Among the included contributions in this study, there were only two approaches that mentioned strangler pattern. Carneiro and Monteiro [24] manually inspected application directories and files. They applied Domain-Driven Design method to define modules, functionalities and boundaries. Then they sorted them by the level of complexity and analyzed their relationships and dependencies. Next, they started to decompose the least complex and small modules into microservices. A new database schema was created in MySQL to interact with migrated microservices. Finally, the developers redirected all front-end components to the new APIs corresponding to the microservices and discarded the legacy functions. These procedures were repeated until the complete microservice application was implemented.

Special care should be taken during migration, Microsoft suggests [14] that system resources, such as database, should be available to both monolithic application and microservices in order to provide functionalities normally. Also, it is recommended to design the new application properly so that they can be further evolved using strangler pattern again. And the

façade should be maintained carefully to avoid becoming a performance bottleneck or failure point. In some situations, strangler pattern is unsuitable. For instance, the application is relatively small and simple, or the user's request to the system is difficult to be redirected.

(4) Continuous Evolution
Continuous evolution aims at iteratively achieving an optimized microservice granularity and service clustering. Several aspects such as quality, level of coupling and cohesion, and functional independence can be inspected and improved by means of multiple times of adjustment of the microservice application. In the end, the result can best meet the expectation for migration.

Eleven contributions in total have included continuous evolution, indicating that this was a popular strategy. Take the approach proposed by Krause et al.[57] as an example, they applied domain analysis, source code analysis and run-time behavior analysis of the legacy application. During this process, Structure101 and ExplorViz were applied. Structure101 was used to inspect source code packages, and it produced maps with a layered structure to bounded contexts and restructured particular packages to solve the conflicts in domain contexts. Therefore, the developers could observe the locations and dependencies of these packages. More importantly, ExplorViz was a trace monitoring and visualization tool for large software. It analyzed the dynamic behavior of the monolithic application to refine bounded contexts. These two tools were available for Java programs. In addition, DBeaver was used to analyze the data model and partition database tables according to respective bounded contexts. Finally, the development teams could further analyze the resulting architecture by examining its run-time behavior, and then they could discover additional microservice candidates. Several iterations were conducted so that the resulting application was refined and could fit the requirement of migration better.

Continuous evolution is a common strategy for the actual migration process. This is supported by the fact that one-third of the final included contributions documented this strategy. It may also be applicable to agile development, and it fits the goal of continuous delivery for producing software updates in short cycles. It allows iterative improvements in production and makes optimized results come true [19][94].

(5) Splitting the Existing Code Base
One of the most popular process strategies was splitting the existing code base. Nearly half of the studies applied this strategy to cut source code and existing monolithic structure to compose a microservice application.

As mentioned in "Required Inputs - Other Inputs" and Technique Type - Workload-Data Aided (WDA)", Jin et al. [47] executed the system with test cases, and they used Kieker to inspect the log files and extracted function-level and class-level execution traces. Next, they clustered the application by Functionality Oriented Microservice Extraction (FoME) method. The Trace Clustering Algorithm was executed to group the classes with the same business logic as one microservice. After this, some shared classes were also extracted as individual microservices for the sake of better maintainability. Forth, they identified interfaces and corresponding APIs for each extracted microservice candidate, and a final loosely-coupled microservice architecture was generated.

This strategy was frequently used because programming languages, imported libraries or packages, and other software resources were usually remained same or similar before and after migration. Therefore, it was unnecessary to switch to new technologies during migration process. Splitting the existing code base according to decomposed architecture design could save costs and efforts, and it could also shorten the time for development and put final microservices into service quickly.

(6) Other Process Strategies
Eight contributions with the research scope applied specific process strategies other than previous mentioned ones. Higashino et al. [42] decomposed the mobile agent system according

to its conceptual diagram. They divided and merged its processes as micro-service candidates according to their network distance to the data to be processed. Data that had strong association (frequency of transactions and communication traffics) with candidate microservices were also partitioned.

On the other hand, Kecskemeti et al. [51] started with ENTICE environment, which was a ubiquitous repository-based technology for virtual machine and container image management. Using this tool and environment, they synthesized use cases and created images accordingly. Next, they broke images into smaller pieces and repeatedly optimized their fragmentation size by analyzing read access operations and microservices' functionalities during test execution. A microservice image family was returned to the ENTICE environment until it reached an optimized status.

What's more, AI-Debagy and Martinek [5] partitioned the APIs and corresponding codes according to name similarity through specific word embedding model analysis.

Last but not least, the remaining contributions mentioned mostly about the decomposition strategies instead of process strategies. Therefore, I assume that splitting the existing code base is usually considered as a default strategy, which is applicable to most cases when it is not specifically mentioned in the contribution. As long as the decomposed design of architecture is completed, in most cases, source code and database can be partitioned accordingly to form a microservice application.

4.2.6 Applicability

In Section 3.2, Fritzsch et al. classified three basic applicability cases: greenfield development (GR), monolith migration (MO) and one that is applicable for both cases (GRMO). These three applicability scenarios will be discussed more detailed in the following paragraph.

(1) Monolith Migration (MO)
Monolith migration (MO), in other words, brownfield development, refers to developing and deploying a new application based on the legacy one. John Wade [105] suggests that this case is usually applicable when developers want to adapt and improve legacy code, integrate new features to the application or enhance its functionality. Relying on existing feasible techniques, it requires less effort and time for development because of the familiar environment and code reuse. However, risks still exist when it comes to performance bottleneck, maintenance cost, unexpected bugs caused by changes, etc. As mentioned by Wade: "Developers are required to obtain comprehensive knowledge about legacy application, including services and data on which they need to develop for the new application. Many parts of existing complex environment need to be re-engineered in order to adjust them to fit the new business requirements and avoid potential failures" [105].

Monolith migration was widely applicable in almost all approaches during my study. Thirty contributions in total presented their migration experience or assumption which were based on monolith migration.

(2) Greenfield Development (GR)
According to John Wade [105], greenfield development (GR) applies a brand-new environment and develops a new application from scratch without any restrictions or dependencies on legacy application. So technological breakthrough and better performance are possible. Nevertheless, higher risk of failure, longer learning curve for new technologies and other negative aspects may also happen in this case.

My study found that six contributions documented their approaches which were related to greenfield development. Their required inputs were usually source code, system specification, UML diagrams, or API specification. Based on these resources, the approach would generate a proper microservice design, which aimed at providing desired business functionalities. New technologies would implement these services.

(3) Applicable for both cases (GRMO)
In my research, I noticed that all greenfield development approaches were also applicable to monolithic migration. Because during greenfield development, a new architectural design indicating the decomposed microservices and their relationships would be provided as a migration result. Therefore, development teams could make trade-offs and choose a suitable case that fits their requirements in the best manner. Apart from exploiting a new environment for migration, developers could also choose to preserve and change the legacy code and environment accordingly. Later they could perform monolith migration as mentioned previously.

After investigation, I concluded that almost every approach was applicable for monolith migration as long as the source code was available in the project. Some contributions were also applicable for greenfield development. This can be decided freely by developers and architects according to their goal, requirements, available cost and so on. Except for these three cases, one contribution focused on quality assessment of microservices, so it was excluded in our discussion.

4.2.7 Validation Type

In order to evaluate the effectiveness of the migration process and the quality of the final product, most contributions documented their validation process after the introduction of their approaches. They provided either experimental evidence, case studies or fictional examples to illustrate how the approach worked in detail.The following paragraph will introduce some specific validation processes recorded in the contributions.

(1) Experiment
Experiment is a validation way that usually selects one or more applications to perform migration according to the steps proposed by the approach. The selected applications are sometimes closed to public users, and they may be developed for personal or experimental use. In other words, the experiment is conducted in order to validate the expected outcome. The experiment is then documented and available to readers to make them understand the methodology of the approach. In my research, I found four contributions which conducted experiment during validation phase.

In the previous paragraph, the approach by Alwis et al. [25] has been introduced already. In order to validate their approach, they migrated two customer relationship management systems: SugarCRM and ChurchCRM. According to Alwis introduction, SugarCRM contained 8116 source files and 101 tables with 600 attributes in total, while ChurchCRM contained 8039 source files and 55 tables with 350 attributes. After migration and optimization, SugarCRM resulted in eight microservices and ChurchCRM resulted in eleven microservices. Next, they measured the "Lack of Cohesion (LOC)" and "Structural Coupling (StrC)" of the legacy systems and migrated systems. These metrics were described by Candela et al.[16]. By comparing the metrics of legacy systems and migrated systems, the improvement of cohesion and coupling of different clusters before and after migration were clearly observed. Then, the legacy systems and migrated systems were deployed in AWS Cloud, and they were executed using Selenium to simulate real-life users. In addition, they also deployed another microservice system, which was decomposed inappropriately, to compare the result with optimized decomposition. Alwis et al. evaluated the effectiveness and performance of the three systems. They observed total execution time, CPU consumption, and network bandwidth consumption during run-time. Besides, they used their own formula to calculate system scalability, availability, and execution efficiency. Finally, they concluded that an optimized microservice system could achieve "higher scalability, availability, efficiency, high cohesion, and low coupling" [25].

(2) Example
To illustrate the migration approach comprehensively, an exemplar and fictional legacy application can be introduced. Usually, this application will have a clear structure without

complex components or inter-relationships, and the migration scenario may also be simple. The researchers can explain the approach step by step in the ideal environment, so readers can easily understand basic methodologies and principles. Validation with example requires less effort, but it also has the risks of lack of empirical evidence and quantitative indicators. So the result of example is convincing than experiment or case study.

In my study, seven research teams described their approach with migration examples. Take Sayara et al. [92] as an example, they described the decomposition approach based on a fictional online reading application, which contained five components: load balancer, catalog service, subscription service, author service, human interface and database. The three main services provided business capabilities to authors and readers. After breaking them down into sub-business capabilities, decomposition algorithm was applied according to each component's update rate and scaling rate. They used Multidimensional Scaling Technique (MDS) to group similar sub-business capabilities together. Finally, this example generated five microservices. Sayara et al. explained each step straightforwardly so that the readers could understand the workflow and apply the approach easily. However, the effectiveness and quality after migration still required further research, since they did not conduct practical validation during their research, such evidence remained vacant.

(3) Case Study
Case study is one of the most effective validation types, because the migration happens in real environment, so unexpected problems and risks tend to expose. Most researchers reported their actual migration projects using the proposed approaches. The advantages and disadvantages of the given approach are also visible after validation. Readers can understand the characteristics of the approach and decide whether they can apply it according to their own scenarios.

Tyszberowicz and his colleagues [103] documented their migration of CoCoME (Common Component Modeling Example). This is a supermarket trading system, and it is often used in case study for software modeling and evolution. The research team used easyCRC tool [102] that assisted word analysis on system specifications about system variables and operations. Then, the TextAnalysisOnline [75] was exploited to visualize a uni-directed bipartite graph and decomposed it so that each candidate has independent system variables and operations. Finally, a partition of the system's state space into microservices was achieved, resulting in four microservices. The RESTish protocols were applied for communication between different services, and each identified microservice was assigned with its own API and database. In order to analyze the changes after migration, the researchers applied KAMP approach [88]. It calculated the changes and its propagation from legacy application to migrated one. Additionally, to validate consistency and correctness of the result, they compared their result with manual decomposition results, which were produced by three student groups. Finally, it was proved that this approach could "provide a decomposition of the system into microservices that was similar to a decomposition suggested by human" [103]. By means of these validations, Tyszberowicz and his colleagues made their approach stand the test of real-life environments. Therefore, it is more convincing to other development teams when they consider referring it.

4.2.8 Tool Support

Another parameter is tool support. This introduces supporting tools for migration which were developed by researchers themselves or from third parties. Some specific tools were already introduced in previous sections. They can be classified into six main categories: analysis tool, monitoring tool, visualization tool, clustering tool (algorithm), development and deployment tool, and management tool.

(1) Analysis Tool
Analysis tool inspects the source code, system specification, database and other resources

to analyze the system's static characteristics. The underlying algorithm varies according to the applicable scenarios. Some tools analyze methods calls among classes and packages, while some tools investigate interface names or words used in the system specifications. For example, FastText or Word2Vec word embedding model analyzes and clusters microservices based on methods' names [5], and Structure101 statically analyzes the source code packages [57].

(2) Monitoring Tool
By means of monitoring tool, the application's run-time behavior can be gathered in various files, such as log files and execution traces. It enables the researcher to investigate events and methods calls that happened during the execution of the target application. Such tools include: Elastic APM [100], openTracing [46], PPTAM+ [46] and so on.

(3) Visualization Tool
Visualization tool is used to generate a graphical representation of the application's structure or behavior. It can provide the researcher an intuitive overview of the application's static or dynamic characteristics. Examples identified during research includes: SArF Map [49], Kibana/ Grafana [46] and DISCO [49].

(4) Clustering Tool (Algorithm)
Rather than manually decomposing the application into microservices, many proposed approaches applied particular tools and clustering algorithms to design the architecture consistently. Usually, the tools can provide "near-optimized" solutions to development teams as reference. For instance, Al-Debagy and Martinek applied affinity propagation algorithms for text analysis and clustering [5], while Selmadji et al. adopted a hierarchical agglomerative clustering algorithm on object-oriented source code [93].

(5) Development and Deployment Tool
Development and deployment tool helps developers to fasten and automate the development and deployment process. It also provides functionalities such as testing, source code merging, library managing, performance monitoring and so on. Such tools mentioned among the contributions includes Junit, Mockito, and Pact for testing [20], Docker for development and deployment [38], JetBrains MPS as a meta-programming framework, etc [13].

(6) Management Tool
Management tool is used for service orchestration during run-time, application maintenance, resource management and so on. According to Mazzara et al. [68], RabbitMQ and Docker Swarm were used for load balancing and service discovery.

4.2.9 Intentions or Quality Metrics Concerned

The final aspect which classified the migration approaches was intentions or quality metrics concerned by the development teams. The intentions indicate drivers or reasons for migration. The microservice migration is usually inspired by the fact that the monolithic application fails to meet new requirements by the development teams, so actions need to be taken in order to enhance further and add business functionalities. Besides, quality metrics are used to evaluate the effects of migration. They represent the quality of final microservices and reflect whether the final results meet the development teams' requirements or not. In my study, These two aspects were discussed together, because both illustrate the desirable characteristics of the microservice architecture to improve their legacy application. In addition, the research question RQ 1.2 in Introduction will be answered in this subsection:

RQ 1.2: What are relevant intentions and quality metrics in a microservice migration/ refactoring scenario?

To classify common metrics as intentions or quality attributes and answer the question, I referred to ISO25010 [44] and defined five factors: maintainability, performance, reliability, scalability, and security. They will be explained sequentially in the following paragraphs.

(1) Maintainability
According to ISO25010 [44], maintainability can significantly influence the secondary developers' experience during software maintenance. Generally, it includes modularity, reusability, analysability, modifiability, and testability. It indicates the degree of effectiveness and efficiency when software maintainers change the application. The maintenance actions include corrections, improvements, and adaption in techniques, environments, requirements and functional specifications by development teams and users.

Maintainability is considered as one relevant parameter here, because microservice architecture to achieve loose coupling and high cohesion. Cojocaru et al. [23] stated that microservice architecture increases the product's delivery rate. It is influenced by granularity, technology heterogeneity, coupling and cohesion. Several matrices can also hint the degree of maintainability, for instance, number of interfaces among microservices, amount of technologies used, etc. Therefore, each service should consider only one specific business functionality. What's more, code data components should be well-separated accordingly. Ideally, changes within one service should be isolated by the boundary as long as the interfaces among services remain unchanged. Other services will be unaware of the changes, they communicate with each other and get results as usual. A good monolith decomposition can improve maintainability and vice versa.

(2) Performance
The performance of software will influence primary users' experience. It is relative to the number of resources used during run-time under specific conditions [44].

About a quarter of the contributions have discussed performance in the validation process. Nakazawa and her development teams [74] utilized visual interface for migration and performance degradation detection. They observed the number of communications between microservices to define performance. Suitable clustering of classes could be realized by reducing unintentional communication frequency, and then the performance can be improved. Also, Ren and his colleagues [87] mentioned duplicating service copies to resolve performance bottlenecks. They conducted empirical research on the influences of different application partitions on performance. Moreover, they also pointed out that frequent communication will cause performance degradation because of network transmission delay. In addition, they measured metrics such as throughput and load before and after the migration. Then they adopted static and dynamic analysis to evaluate performance improvement. Even more, Mahanta and Chouta [64] imported C packages to evaluate performance by observing and instrumenting code flows. Besides, Alwis et al. [25] applied virtual machine technology to host microservices in AWS Cloud to record and compare execution time, CPU consumption and network bandwidth consumption to refine the performance. Further, Three contributions [23][46][38] evaluated performance by response time and throughput, which could be measured at run-time using various tools such as PPTAM+ [46].

In my study, performance can be related to many sub-parameters. According to [3], performance efficiency includes time behavior, resource utilization and capacity. This is one of the standards that the development teams can refer to during the validation process, and they can select and define their own metrics that fit their requirements.

(3) Reliability
Reliability is also a widely discussed parameter in the study. It should be concerned by both users and development teams including stakeholders [3]. ISO25010 [3] defined reliability as the ability of a system to provide specific functions under specified conditions within the predefined duration. It is a combination of maturity, availability, fault tolerance and recoverability. The microservice application should be accessible by the user under a certain

level and it should meet users' requirements under regular operation. It should also bear specific hardware or software failures and re-establish the desired state within a certain period of time.

Cojocaru explained the concept of Service Level Agreements (SLA) [23], which is a multiplicative quality attribute describing the availability over a certain period of time. A lightly change of SLA will result in a dramatic increase or decrease of the overall availability of the application.

Specifically, Mazzara et al. [68] admitted that microservice architecture can provide better availability. Because it is loosely coupled and it can apply replication and load-balancing for individual services. Also, with the help of RabbitMQ and container technology, independent microservice environments and reliable message transmission are easily realized to enhance availability. Finally, Alwis and his colleagues [25] proposed Non-dominated Sorting Genetic Algorithm (NSGA) II for migration. They evaluated the availability with two customer management systems by measuring service up time and response time. They then calculated optimal values to make trade-offs between availability, scalability, granularity, and execution cost. There were some contributions that mentioned other metrics to determine reliability. For example, Carneiro and Monteiro [24] discussed the usage of different frameworks and tool-kits to improve system resilience, while Taibi and Systä [100] claimed that microservice architecture might also enhance fault-tolerance.

Reliability was also frequently discussed in the contributions because microservice architecture encourages separate business capabilities and independent service development based on different environments and technologies. In that case, it brings side effects such as greater redundancy and isolation of failure within service boundaries. The nature of microservices provides potential enhancement of reliability in various ways.

(4) Scalability
Scalability is a parameter that was added specifically in my study. O'Brien [77] defined scalability to be the ability to provide correct functionalities with the same performance regardless of the changes in application size or amount of resources. He classified it into horizontal scalability, which meant duplicating the microservice; and vertical scalability, meaning adding the amount of resources to a microservice. Cojocaru [23] further measured it by calculating the distribution of requests provided by services under various workloads.

According to the study by Mazzara et al. [68], microservice architecture can increase system scalability because of reduced complexity, low coupling, high cohesion, and simple integration. All sub-parameters, including automation, orchestration, service discovery, load balancing, and clustering should be carefully maintained. They suggested applying platforms such as Google Kubernetes [2], Mesosphere Marathon [1] and Docker Swarm Mode [4] to enhance scalability. Additionally, Fan and Ma [20] stated that the architecture should also integrate API gateway for service invocation, and circuit breaker as error handling mechanism.

As mentioned in "Technique Type - Meta-Data Aided (MDA)", Sayara [92] also defined and decomposed business capabilities according to the scaling rate of the monolithic system. They calculated the scaling matrix for each business capability and applied algorithms to group similar capabilities as one microservice. They stated that microservice application ensured independent development, bug isolation, and accurate scaling of desired resources.

What's more, according to the precious approach by Alwis and his team [25], they proposed Non-dominated Sorting Genetic Algorithm (NSGA) II, aiming at achieving microservices according to the factors such as scalability cost. They claimed that "a system is well scalable if it can provide services with less time and fewer resources in an under-provisioned state". So they measured the time and resources taken by the services to quantify scalability cost. Then they used it as one major factor of their decomposition algorithm and generated a set of clustered business objects and operation nodes. Also, Tyszberowicz, Heinrich and Liu [103] said that a fine-grained microservice clustering can also improve scalability.

In the end, sixteen contributions have evaluated or mentioned scalability as one important benefit of microservice migration. Considering the fact that many worldwide Internet businesses and services, such as Netflix and Amazon, have already applied microservice architecture to handle the huge website traffics and continue offering good user experience, it makes sense that scalability should also be paid attention to for migration intention or quality assessment.

(5) Security
Security is also one quality metric under consideration during this study. Because microservice architecture will usually separate independent service modules on different hosts, and they will communicate with each other by interfaces, this may arise additional risks during the communications via network. Therefore, it is curious to researchers how microservice architecture can ensure certain security level during their design and implementation process.

ISO25010 [3] said: "Security is important to user experience. It consists of confidentiality, integrity, non-repudiation, accountability and authenticity. The system should be resilient to malicious attacks in certain level and continue providing services to authenticated users."

Cojocaru et al. [23] conducted a detailed investigation on security. In their research, they found that the security of microservice still lacked reliable testing methodologies. One feasible way was to inspect execution traces and UML diagrams according to domain models. They also suggested that the security was not ensured by microservice architecture. Instead, it depended on technologies and some particular aspects of the application [23]. Therefore, they made one conclusion that microservice architecture provided one possibility to isolate vulnerabilities such as sensitive data to improve security. Additionally, Maisto and his team [65] aimed at realizing DevOps practices for migration process in combination with the latest security standards.

There were only two contributions in my study that mentioned security as migration intention or validation aspects. There were few formal methodologies to validate security. This may because microservice architecture, especially migration, is still a "young" topic under research. Further study about this scope is desirable to find more formal and various validation evidence about it.

(6) Other Intentions or Quality Metrics
Apart from the intentions and quality metrics listed above, some other parameters were introduced in different contributions.

One of the most frequently mentioned parameters was cost and effort reduction [91][68][23]. Because of the better structure of the microservice architecture, the maintenance was easier for the development teams. It required less effort to add new features and make changes. The automation of development and deployment also made the procedure more agile and cost-effective [20][24][66].

In addition, Taibi and Systä [100] suggested that microservices can reduce the need of interaction between teams. Because the interfaces for inter-service communication may be already defined and development teams can focus on high cohesive service with one business functionalities, they can spend all their effort on their own development and require lower support from others. This is supported by Gouigoux and Tamzalit [38], as they said that microservice architecture could increase financial return over investment by reducing the need for communication between teams.

What's more, Mazzara et al. [68] explained that they transformed the monolithic application to microservices to achieve a distributed and modular application. Knoche and Hasselbring also stated [26] that such structure could reduce complexity and avoid vendor or technology lock-in in the future. Besides, Tyszberowicz and his teams [103] migrated a monolithic application to achieve better traceability.

According to the result of my study, it is obvious that maintainability and scalability are the two main parameters considered by development teams and architects throughout the migration. They also reflect the advantages and benefits provided by such modern architecture. Microservices are loosely coupled services that can be deployed on independent hosts, so maintenance becomes easier and faster. Once the resources are insufficient to provide services to the users, they can scale easily by deploying more hosts and resources accordingly to ensure the quality of services (QoS). Other parameters such as better performance, higher reliability and security are frequently considered as well. From my perspective, the increase of quality is one of the actual benefits of microservice architecture, and it simulates the architects and developers to migrate their legacy code. This part can be investigated to find more related parameters that people care about when migrating their monolithic applications.

After all included contributions were read thoroughly, a five-point likert score was assigned to each contribution. As I mentioned before, it indicated the quality, comprehensiveness, weight of importance and feasibility of contributions. Next, a data extraction form was created for each contribution, and the relevant information was extracted and filled into the form in a predefined format, which can be seen in the column "Value" in Figure 3.1. In this case, I could reduce effort for rework during data synthesis because it was considered one of the largest wastes in software engineering. As a result, a repository of Excel files containing all required data of migration approaches was achieved.

4.3 Contribution Selection

After my full-text reading was finished, Fritzsch and I performed contribution inclusion and exclusion again to to refine the field of study further. Then a primary contribution index was created accordingly. It contained information about all sixty-one contributions, including title, database source link, decision of inclusion or exclusion, note, etc. The content of the index is shown in Appendix A Table A.1. Moreover, we further classified the included contributions by three main aspects:

(1) Approach/ Framework/ Review
These aspects indicated whether the authors proposed an approach, a framework for migration, or they conducted a review of several migration projects.

(2) Validation Type
The validation type was already introduced in the previous section. Additionally, the contribution with ID "13" conducted several studies to measure the quality of various migrated applications. Therefore, I classified it as "Meta Study". I kept it in the repository since quality was also one of the essential aspects for migration and it also represented the intention for migration. Besides, it was well documented, so readers could quickly grasp the concept by reading the content.

(3) Process-related/ Tool-related/ Quality Assessment
These aspects specified the main focus of the contribution. The authors might mainly illustrate detailed steps and techniques used for migration, or they might introduce particular tools aiding the structural analysis, decomposition, migration, deployment and so on. Besides, the authors might conduct various validation processes to evaluate and quantify quality metrics, so the effectiveness and quality of the migration approach or framework were proved.

Based on two researchers' selection, a final list of included contributions was achieved. This list was used to answer the research question RQ 1:

> *RQ 1: What approaches or frameworks for migration scenarios are proposed in scientific literature?*

In the end, thirty-one contributions were selected after full-text reading, this can be seen in Table 4.1. Moreover, the final included contributions and their corresponding type information are available in the following Table 4.2.

Table 4.2: Final Included Contributions and Type Information

ID	**Title**	**Approach/ Framework/ Review**	**Validation Type**	**Process-related/ Tool-related/ Quality Assessment**
1	From monolith to microservices: Lessons learned on an industrial migration to a Web Oriented Architecture [38]	Approach	Case Study	Quality Assessment
2	A Probabilistic Approach For Obtaining An Optimized Number Of Services Using Weighted Matrix And Multidimensional Scaling [92]	Approach	Example	Process-related
3	Microservices architecture: Case on the migration of reservation-based parking system [108]	Approach	Case Study	Process-related
4	Functionality-oriented Microservice Extraction Based on Execution Trace Clustering [47]	Approach	Case Study	Process-related
5	Transform Monolith into Microservices using Docker [91]	Approach	Example	Process-related
7	Visualization Tool for Designing Microservices with the Monolith-first Approach [74]	Approach	Case Study	Process-related + Tool-Related
9	Microservices: Migration of a Mission Critical System [68]	Approach	Case Study	Tool-Related
11	Extracting Candidates of Microservices from Monolithic Application Code [49]	Approach	Case Study	Process-related + Tool-Related
13	Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications [23]	Review	Meta Study	Quality Assessment
14	Automatic performance monitoring and regression testing during the transition from monolith to microservices [46]	Approach	Example	Quality Assessment
26	Migrating Web Applications from Monolithic Structure to Microservices Architecture [87]	Approach	Experiment	Process-related
31	A logical architecture design method for microservices architectures [90]	Approach	Case Study	Process-related
32	Availability and Scalability Optimized Microservice Discovery from Enterprise Systems [25]	Approach	Experiment	Process-related
34	A Model-Driven Approach Towards Automatic Migration to Microservices [13]	Approach	Example	Process-related + Tool-Related

Continued on next page

Table 4.2 – *Continued from previous page*

ID	Title	Approach/ Framework/ Review	Validation Type	Process-related/ Tool-related/ Quality Assessment
39	Towards a Methodology to Form Microservices from Monolithic Ones [51]	Approach	Not specified	Process-related
40	Translating a Legacy Stack to Microservices Using a Modernization Facade with Performance Optimization for Container Deployments [64]	Framework	Example	Process-related
41	From Monolith to Cloud Architecture Using Semi-automated Microservices Modernization [65]	Approach	Not specified	Process-related
42	From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [76]	Approach	Case Study	Process-related + Tool-Related
43	Tool Support for the Migration to Microservice Architecture: An Industrial Case Study [82]	Approach	Case Study	Process-related + Tool-Related
44	Re-architecting OO Software into Microservices A Quality-Centred Approach [93]	Approach	Case Study	Process-related + Tool-Related
45	An Experience Report from the Migration of Legacy Software Systems to Microservice Based Architecture [24]	Approach	Experiment	Process-related
47	A Reconfigurable Microservice-Based Migration Technique for IoT Systems [97]	Approach	Case Study	Process-related
48	Identifying Microservices Using Functional Decomposition [103]	Approach	Case Study	Process-related + Tool-Related
50	Microservice Decomposition via Static and Dynamic Analysis of the Monolith [57]	Approach	Case Study	Process-related + Tool-Related
53	A Design with Mobile Agent Architecture for Refactoring A Monolithic Service into Microservices [42]	Approach	Example	Process-related
55	From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining [100]	Framework	Case Study	Process-related
56	A New Decomposition Method for Designing Microservices [5]	Approach	Case Study	Process-related
57	Migration of Software Components to Microservices: Matching and Synthesis [22]	Framework	Experiment	Process-related + Tool-Related
58	Use Case Driven Microservices Architecture Design [66]	Approach	Example	Process-related + Tool-Related
60	Using Microservices for Legacy Software Modernization [54]	Approach	Case Study	Process-related
61	Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report [31]	Approach	Case Study	Process-related + Tool-Related

After the second contribution selection, effort required for study was greatly reduced because of narrower research field. Also, the quality of each contribution was enhanced and its relevance to my study topic was also ensured. A contribution index with a clear structure and basic information could also provide me an easier way to manage the contributions and future benefited the set up of repository for the web-based tool.

However, one problem was identified that only three out of thirty-one contributions introduced migration frameworks. Mahanta and Chouta [64] proposed a framework for the application's architectural evaluation and simulation. They followed specific steps to analyze the application by URL and source code, and then the framework tried to decompose the application and optimize it by simulation. Also, Christoforou et al. [22] designed a migration framework using EBNF Profiles. The framework used ontology alignment algorithm to group similar components and generate a microservice architecture design. Lastly, Taibi and Systä [100] introduced a six-step framework to perform decomposition based on execution log files. Their framework provided several decomposition alternatives for optimized approaches decomposition, and developers could choose one design according to their requirement. Among these three frameworks, only Taibi and Systä [100] proposed an explicit and formal framework because its applicable scenario was generic and it generated multiple solution choices for the user. The other two frameworks [64][22] claimed by the authors were relatively similar to an approach designed for particular scenarios. The vague or mixed definition of an approach and a framework among some contributions provided difficulties in classifying them during full-text reading, and it might affect the correctness of the result. Therefore, special care needed to be taken during the review to eliminate such confusion.

4.4 Data Synthesis and Framework Definition

Finally, data synthesis was done to make the extracted data consistent. The number of contributions mentioning specific elements in the data extraction form was already described in Section 4.2. Additionally, the statistical data of some essential elements are shown in the following pages. Based on these figures, an overview of the whole repository is clear to the readers, and it is easy to figure out which strategy or technique is most frequently used by the researchers. Please note that one contribution could apply multiple techniques or strategies in their approach or frameworks, so the sum of numbers in each figure can be greater than the total number of contributions.

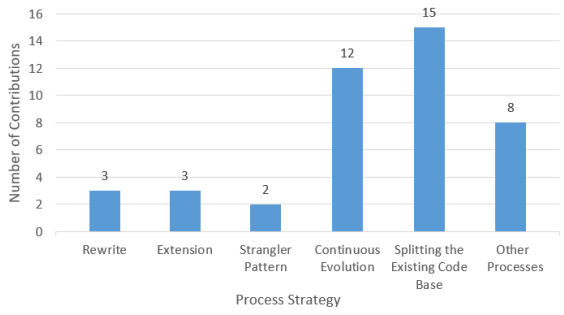


Figure 4.6: Statistic for Process Strategy

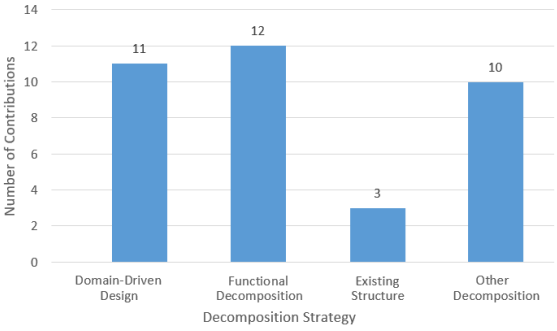


Figure 4.7: Statistic for Decomposition Strategy

Based on the knowledge acquired from the literature review, a microservice migration framework was designed. The basic concept came from Fritzsch and his colleagues' study [33]. They proposed a decision guide that acted as a migration framework. According to user's requirement, available resources and system structure, they could provide migration advice about what kind of technique types should be applied (SCA, MDA, WDA, DMC). However, it only focused on technique suggestions instead of other aspects such as process strategy, decomposition strategy,

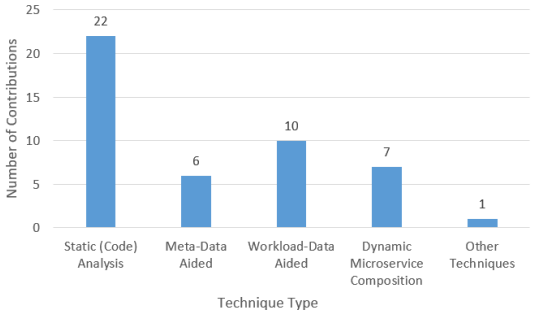


Figure 4.8: Statistic for Technique Type

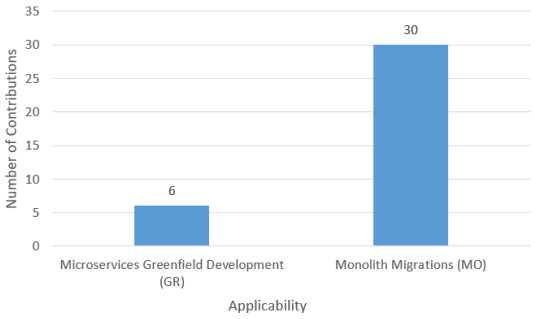


Figure 4.9: Statistic for Applicability

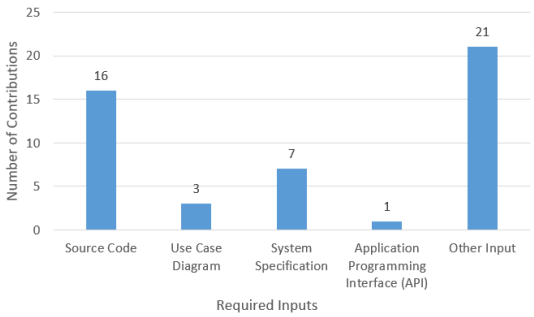


Figure 4.10: Statistic for Required Inputs

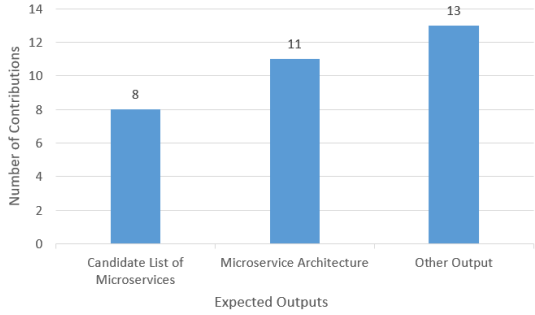


Figure 4.11: Statistic for Expected Outputs

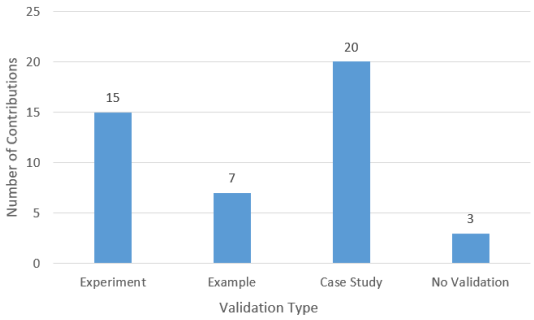


Figure 4.12: Statistic for Validation Type

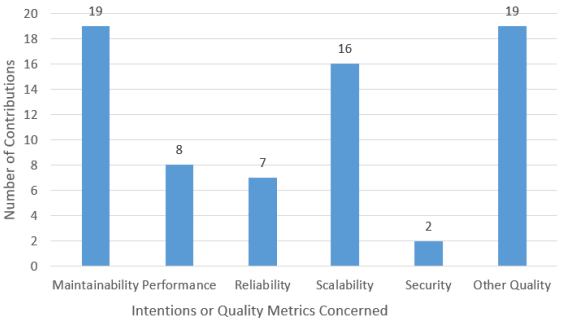


Figure 4.13: Statistic for Intentions or Quality Metrics

tool support and so on. So I adapted their framework and extended it further into a more generic one. As we can see in Figure 4.14, the framework takes several inputs from users, which were particularly introduced in previous paragraphs. These are the essential parameters required by the framework in order to suggest possible migration approaches referring to the included contributions in the repository. The suggested results should be arranged in the order of relevance. The high relevance indicates that this contribution provides an approach or framework which is more suitable to user's requirement. In that case, users have a set of choices, and they can study different migration approaches or frameworks and select one which fits their scenarios most.

Till now, the systematic literature review and framework design were completed. A general overview of the current state of research about architectural refactoring was acquired. Also, a repository of data extraction forms were generated containing all relevant knowledge about microservice migration approaches and frameworks. Therefore, I had a solid foundation to further design and

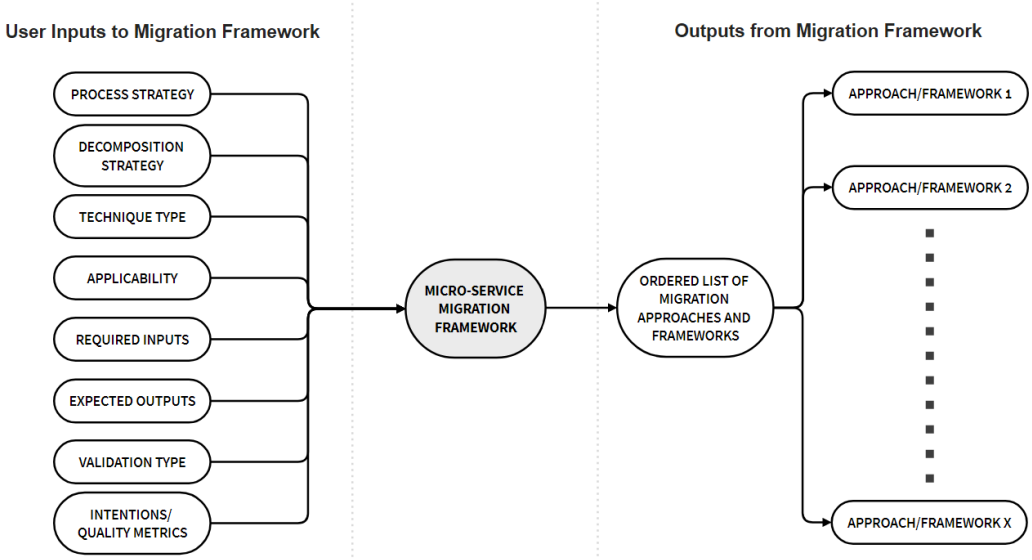


Figure 4.14: Design of Microservice Migration Framework

implement a web-based tool to guide the choice of refactoring techniques and approaches.

Chapter 5

Web-based Tool Design and Implementation

In this chapter, I will finally the answer research question RQ 2, that is:

> *RQ 2: How can we design a tool to serve the architects and developers as guidance for microservice migration?*

In order to present and aggregate the knowledge acquired from the literature review, a web-based tool was designed and implemented. Its main objective was to help software architects and developers overlook and comprehend the existing research status and help them choose suitable refactoring techniques and approaches. The repository produced by the literature review could be transformed into an actual database containing all relevant properties of the analyzed refactoring approaches. Furthermore, it should be connected to the web-based tool to provide such functionality to users. In this project, several techniques were used to implement the tool. The front-end (user interface) was developed using Bootstrap, HTML, PHP, and JavaScript; the back-end (database) was set up using XAMPP and MySQL; the communication between front-end and back-end was realized mainly by PHP.

5.1 Use Case Diagram

In order to identify the use cases of the web-based tool, the use case diagram was first designed. It represented the things that users could do with this tool. As we can see in Figure 5.1, I defined five main use cases, those were: search for approach/ framework, read search result list, read detailed approach information, select scenarios, and get original contribution.

(1) Search for Approach/ Framework
In index page, the user can search for approaches or frameworks according to the user's input from the user interface (UI), the input should represent the characteristics of user's monolithic application, the user's expectation and requirement to the new microservice application, which were discussed in Section 4.2.

(2) Read Search Result List
After searching, the user can go to search page and read a result list containing all approaches or frameworks which are recommended by the tool and suitable to his/ her monolithic application. They are well-ordered by ID, year, or degree of relevance. The order can be decided by the user.

(3) Read Detailed Approach Information
If the user click one title in the result list, the tool will open a new detailed result page and show detailed knowledge about the migration process within its specific scenario.

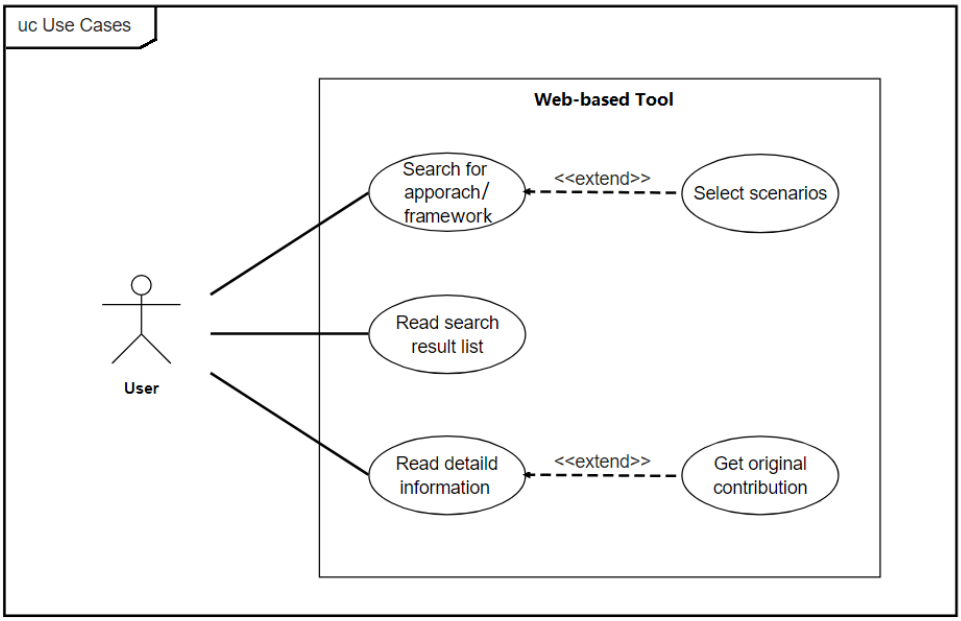


Figure 5.1: Use Case Diagram of the Web-based Tool

(4) Select Scenarios In index page, the user can select parameters in each property, these properties were discussed in Section 4.2. After selection, the user can click "Search" button to perform searching. Additionally, he/ she can read all migration approaches or frameworks by clicking "Show All" button without selecting any parameters.

(5) Get Original Contribution
In result page, the user can also click a link to jump to the original source website for further information and retrieve particular contribution source file.

5.2 Functional Requirements

Several main functionalities were defined during the design phase of the tool. They specifically realized the use cases as mentioned before, and they are listed and described as follows:

(1) Characteristic Selection
Via a web-based tool UI, the user can specify a set of system characteristics according to his/ her specific application, referring to the scenarios I have introduced in Section 4.2. In that case, the web-based tool can be acknowledged about the most important and prior characteristics that user cares about, and use them to search for relevant contributions suitable to user's requirement.

(2) Search Button
The web-based tool should conduct SQL searching mechanisms to the database via internet connection. After the user clicks "Search" button in index page, an SQL search string is generated according to the user's input. The web-based tool should be able to achieve the corresponding results after searching.

(3) Show All Button
The web-based tool should allow the user to skip characteristic selecting and view all contri-

butions in the database. After the user clicks "Show All" button in index page, the web-based tool should achieve all information from the database.

(4) Show Results
After searching, the user goes to search page, and the web-based tool will provide a list of relevant results, containing columns such as ID, title, year of publication, author, missing characteristics of the contribution, quality score and recommendation score in percentage.

In detail, the contribution may not contain all desired characteristics by the user. So the missing characteristics of the contribution in the column shows what characteristics are not included in one specific contribution. Besides, the recommendation score represents the degree of relevance and initial quality score of the contribution. It is calculated by a specific equation, which will be discussed in Section 5.5. Therefore, the user can refer to these two columns to check if the contribution is suitable and relevant to his/ her requirements.

(5) Result List Sorting
Additionally, there are four buttons in search page: "IDSort", "YearSort", "Score" and "RelevanceSort". The tool provides user possibility to order the result list by ID, year of publication,the quality score of contribution or recommendation score. By clicking the button, the list can be sorted in ascent or descent order automatically. This enables user to scan the most interesting result.

(6) Read Detailed Approach or Framework Information
By clicking the title of one contribution in the result list, the web-based tool will open a new detailed result page, where all detailed information, such as process strategy, decomposition strategy, required inputs, expected outputs, technique steps, and tool support is available to the user. So the user can be informed of all technical details about the proposed approach or framework.

(7) Go to Source Page
In result page, the user can click "Link to Website" button, and the tool will open a new page to jump to the source database website. So he/ she can read original information about the contribution and retrieve the source file if he/ she has membership or license for the database.

(8) Instruction
In index page, the tool should show a link to the instruction page, so that the user can understand how to use the tool. Here, a GitHub README page [40] is suitable to provide such instruction.

5.3 User Interface

After the functionalities of the tool were defined, the mock-up of the UI was designed. It contained all UI elements as mentioned before and made functionalities visible to the user. Three pages were designed for the tool, including index page, search page and result page.

According to Figure 5.2, the index page contains various sections representing user's monolithic application's characteristics and his/ her requirement to the new microservice application. These sections were already discussed in Section 4.2.

Initially, I only created two radio buttons for each property in the sections: "Include" and "Exclude" button. However, it is possible that the user does not care about particular properties. For instance, he/ she does not care about what kind of input types are required for migration, so it is confusing to the user to decide inclusion or exclusion in the "Input" section. Therefore, I later changed it into three radio buttons: "Include", "Exclude", and "Neutral". So the user can decide whether to include this property, or exclude it, or leave it as neutral, meaning that it does not matter if the contribution has this particular property or not. The choices will influence the SQL search string and produce a corresponding result list. This will be mentioned again in Section 5.5.

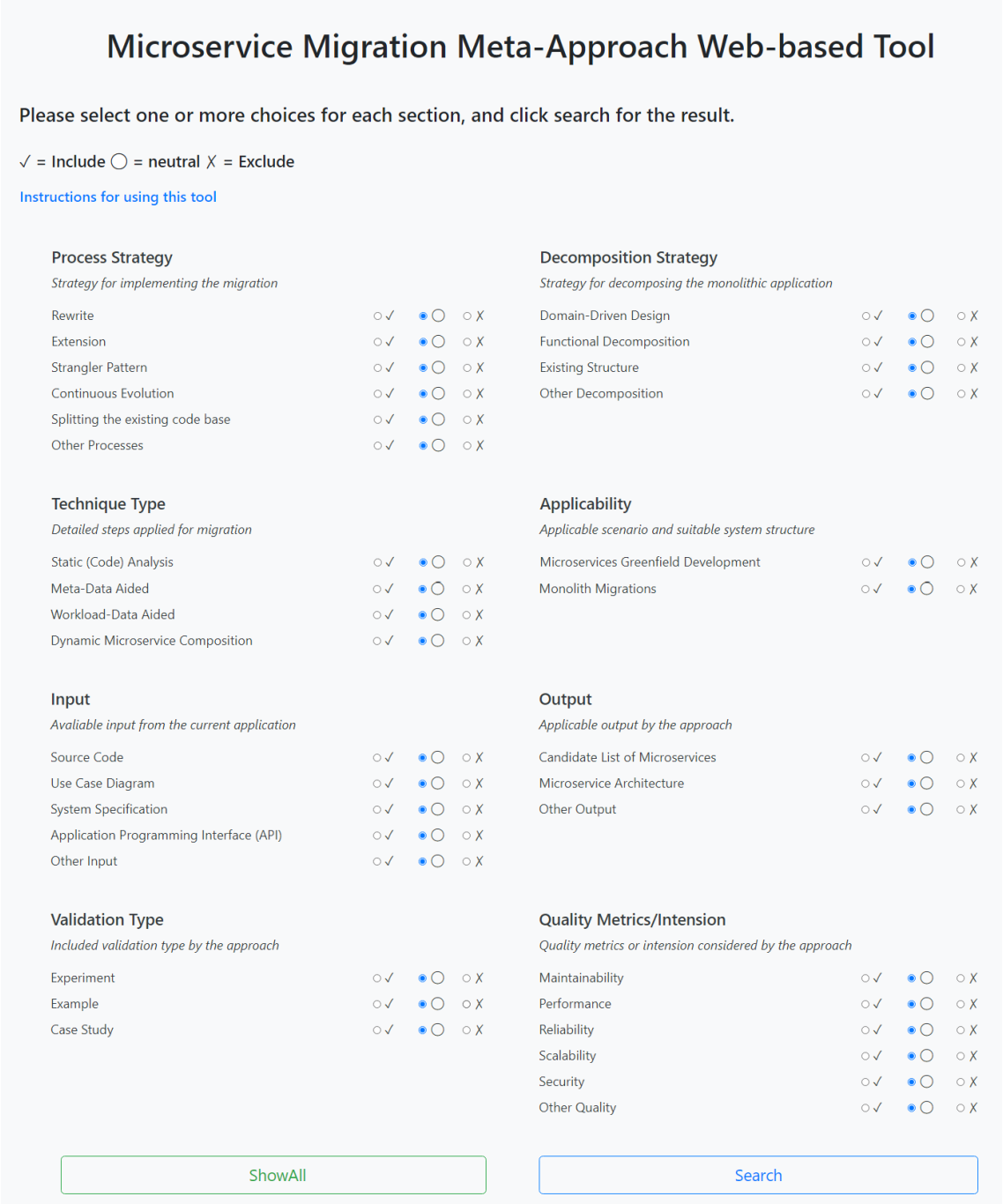


Figure 5.2: The Index Page of the Web-based Tool

Figure 5.3: The Hover Effect by Bootstrap in Index Page

Also, in order to instruct the user and provide enough hints about usage, each section title has hover effect. As we can see in Figure 5.3, if the user moves his/ her mouse cursor onto the title of a section, a description will pop up to explain the terminology and concept of this section. This can be realized easily using the Bootstrap framework. And the code example is provided as follows:

Listing 5.1: Bootrap Bover Effect Definition

```
1 <h5 data-toggle="tooltip" data-placement="bottom" data-delay='{"show":"800"}' title="Some description
      example">
2 Title of Section
3 </h5>
```

Bootstrap enables developer to easily realize hover effect by simple setting the "data-toggle", "data-placement", and "data-delay" attributes in HTML-5. Here, a plugin called Tooltips was used in "data-toggle". According to the official tutorial: "it relies on the 3rd party library Popper.js, so developers must include popper.min.js before bootstrap.js, or use bootstrap.bundle.min.js/ bootstrap.bundle.js" in order to make use of Tooltips [12]. According to the code block example, I set hover text to show at the bottom of the title, and I set the hover effect time delay to be eight hundred milliseconds. Therefore, it can prevent immediate popping up and disturbing the user.

After the user click "Search" or "Show All" button in index page, the tool will provide a list of relevant results in search page. The layout is shown in Figure 5.4. As mentioned before, a search string showing the selected properties by the user is again visible to him/ her as feedback. Additionally, the total number of results is shown above the result list.

The result list has toggling row color for the sake of better readability. And user can click sorting buttons to sort the list in different orders. Specifically, data in the "Missing String" column indicates the missing characteristics desired by the user. They are shown in HTML Strikethrough element, using "<del>" tag, so the user can be straightforwardly informed about the missing properties of each proposed approach or framework.

What's more, in the upper left corner of the page, there is a link called "Back to Index Page", the user can click it to close the search page and return to the previous index page easily.

Finally, if the user clicks one title in the search page, he/ she can jump to a new result page. Here, all collected data of an approach/ framework is visible in a well-designed table layout. The user can read the migration information, including "Process Strategy", "Decomposition Strategy", "Technique Type", etc. This page can provide the user with a guideline for architectural refactoring, and he/ she can decide whether it is applicable to his/ her monolithic application.

Additionally, in the upper right corner of the page, there is a blue button called "Link to Website", the user can click it and jump to the corresponding database for further details and source file.

Generally speaking, the main focus of this tool was to provide practical usage and valuable information gathered from literature review, so less effort was paid on fancy animations or effects. All UI elements and design mainly served to provide basic and necessary functionalities to users in order to support easy and convenient usage of the tool.

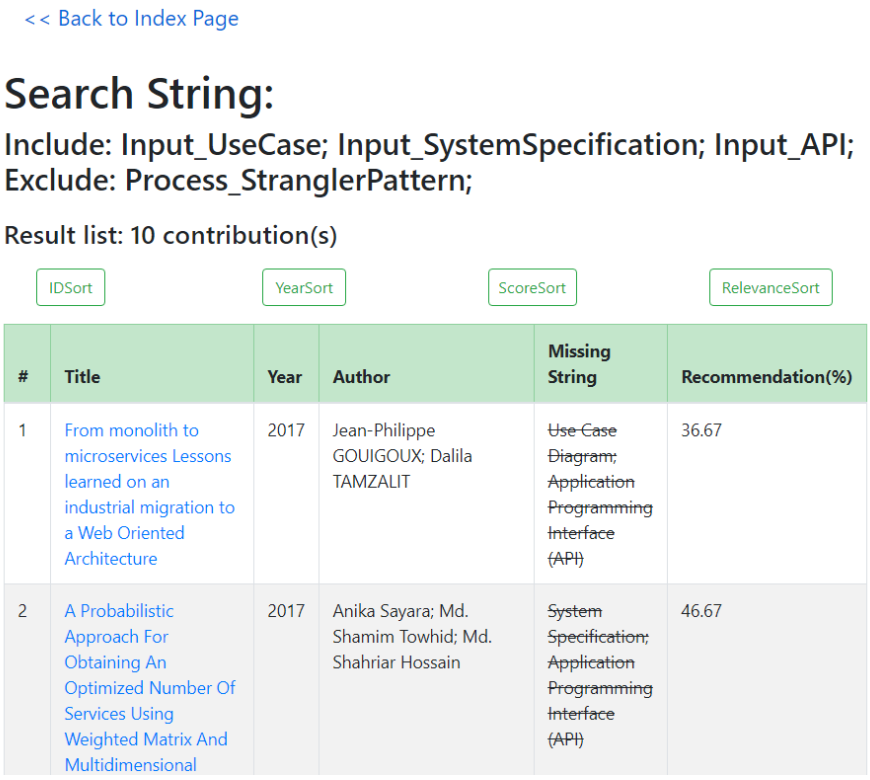


Figure 5.4: The Search Page of the Web-based Tool

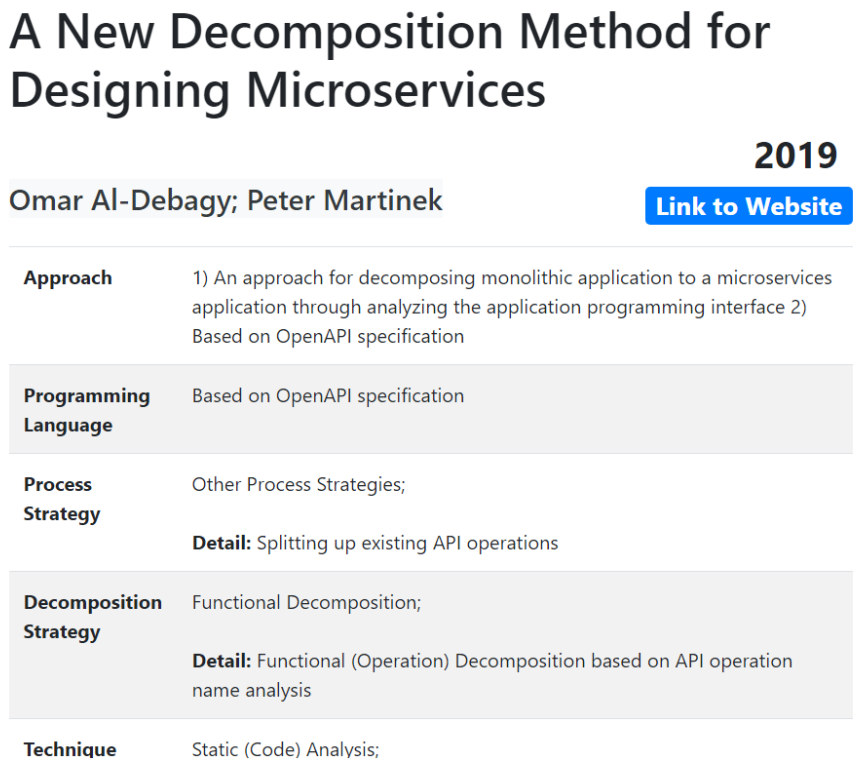


Figure 5.5: The Result Page of the Web-based Tool

5.4 Database Structure

After designing the UI, the next step was to set up the web-based tool database. In this step, the XAMPP was set up on the computer for local internet application development. As introduced on the official website [6], it is an open-source PHP development environment. And it has built-in phpMyAdmin MySQL database administration tool, Apache cross-platform web server software, and other software components for easy installation and local development.

With the help of phpMyAdmin, a MySQL database schema "migration" and a table "contribution" were created for the tool. The basic structure of the table is available in the following code block. All columns were defined according to the elements in index page. These reflected the framework proposed by me, and they were considered as the most essential parameters during architectural refactoring and microservice migration. Most columns related to essential parameters were defined as Boolean type ("tinyint(4)")and Not Null. Except for the basic information, such as ID, title, year of public cation, authors, and some descriptive data such as process detail, technique sets for migration, were defined as medium text field or variable character field. Therefore, primary validity and space usage were ensured. The database columns were named consistently so that it could be easily figured out which column belonged to which scenario. For example, the column "Process_Rewrite" indicates if the proposed approaches or frameworks are related to rewrite pattern for process strategy.

Listing 5.2: The Structure of Table "Contribution"

```
CREATE TABLE `contribution` (
`id` int(11) NOT NULL,
`Title` varchar(150) NOT NULL DEFAULT 'Title',
`Year` year(4) NOT NULL DEFAULT 2000,
`Authors` tinytext NOT NULL,
`Link` varchar(250) NOT NULL DEFAULT 'Empty Link',
`Approach` text NOT NULL,
`Process_Rewrite` tinyint(4) NOT NULL DEFAULT 0,
`Process_Extension` tinyint(4) NOT NULL DEFAULT 0,
`Process_StranglerPattern` tinyint(4) NOT NULL DEFAULT 0,
`Process_ContinuousEvolution` tinyint(4) NOT NULL DEFAULT 0,
`Process_Split` tinyint(4) NOT NULL DEFAULT 0,
`Process_Others` tinyint(4) NOT NULL DEFAULT 0,
`ProcessDetail` mediumtext DEFAULT NULL,
`Decomposition_DDD` tinyint(4) NOT NULL DEFAULT 0,
`Decomposition_FunctionalDecomposition` tinyint(4) NOT NULL DEFAULT 0,
`Decomposition_ExistingStructure` tinyint(4) NOT NULL DEFAULT 0,
`Decomposition_Others` tinyint(4) NOT NULL DEFAULT 0,
`DecompositionDetail` tinytext DEFAULT NULL,
`TechniqueSet` mediumtext NOT NULL,
`Technique_SCA` tinyint(4) NOT NULL DEFAULT 0,
`Technique_MDA` tinyint(4) NOT NULL DEFAULT 0,
`Technique_WDA` tinyint(4) NOT NULL DEFAULT 0,
`Technique_DMC` tinyint(4) NOT NULL DEFAULT 0,
`Technique_Others` tinyint(4) NOT NULL DEFAULT 0,
`TechniqueTypeDetail` mediumtext DEFAULT NULL,
`Applicability_GR` tinyint(1) NOT NULL DEFAULT 0,
`Applicability_MO` tinyint(1) NOT NULL DEFAULT 0,
`AtomicUnit` tinytext DEFAULT NULL,
`Input_SourceCode` tinyint(4) NOT NULL DEFAULT 0,
`Input_UseCase` tinyint(4) NOT NULL DEFAULT 0,
`Input_SystemSpecification` tinyint(4) NOT NULL DEFAULT 0,
`Input_API` tinyint(4) NOT NULL DEFAULT 0,
`Input_Others` tinyint(4) NOT NULL DEFAULT 0,
`InputDetail` tinytext DEFAULT NULL,
`Output_List` tinyint(4) NOT NULL DEFAULT 0,
`Output_Archi` tinyint(4) NOT NULL DEFAULT 0,
`Output_Others` tinyint(4) NOT NULL DEFAULT 0,
`OutputDetail` tinytext DEFAULT NULL,
`Tool` mediumtext DEFAULT NULL,
`ProgramLanguage` mediumtext DEFAULT NULL,
`Validation_Experiment` tinyint(4) NOT NULL DEFAULT 0,
```

```
43 `Validation_Example` tinyint(4) NOT NULL DEFAULT 0,
44 `Validation_CaseStudy` tinyint(4) NOT NULL DEFAULT 0,
45 `Validation_NoValidation` tinyint(4) NOT NULL DEFAULT 0,
46 `ValidationTypeDetail` mediumtext DEFAULT NULL,
47 `ValidationMetrics` mediumtext DEFAULT NULL,
48 `Quality_Maintainability` tinyint(4) NOT NULL DEFAULT 0,
49 `Quality_Security` tinyint(4) NOT NULL DEFAULT 0,
50 `Quality_Performance` tinyint(4) NOT NULL DEFAULT 0,
51 `Quality_Reliability` tinyint(4) NOT NULL DEFAULT 0,
52 `Quality_Scalability` tinyint(4) DEFAULT 0,
53 `Quality_Others` tinyint(4) NOT NULL DEFAULT 0,
54 `QualityDetail` mediumtext DEFAULT NULL,
55 `Score` float NOT NULL DEFAULT 1
56 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

The database data for each row were entered from the data extraction forms of included contributions. This was done manually, and spelling errors or format errors were corrected again in this step. Because of the formal data extraction form, this process was done quickly. In addition, a user with only read privilege was created to access the database in order to protect data from malicious manipulations or unintended changes. Besides, the developers could update, add or delete data using phpMyAdmin administration tool directly with a web browser, having the full privileges for operations. The SQL sentence example for creating a user in the local database is shown as follows:

Listing 5.3: SQL Sentence Example for User Creation

```
1 CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';
2 GRANT SELECT ON migration . contribution TO 'user'@'localhost';
```

After a well-structured and well-named database table was established, I could easily develop the searching and matching algorithms later. The access to the database from local web page was possible using Apache in XAMPP, and the detailed setup process of the local environment was described in GitHub README page [40].

5.5 Programming and Class Diagram

Based on the design of UI and database, the web-based tool was finally developed, providing actual functionalities to user. The class diagram in Figure 5.6 graphically depicts the whole structure of the web-based tool.

As mentioned before, I designed three web pages for the tool: index, search, and result. They were designed with HTML in PHP file, which is shown in the figure. All of them utilized external scripts and css files, including "pooper.min.js", "bootstrap.min.js", "bootstrap.min.css", and self-developed "style.css". They could realize some animation effects such as hover text and implement UI elements easily with pretty appearance.

The index page contains mostly UI elements and a few methods. The main purpose of index page is to collect user's input. After the user clicks "Search" button, it will pass all input elements value to the new search page to retrieve the corresponding result from the database. Additionally, if the user clicks "Show All" button, all contributions in the database would be listed in search page, regardless of user's inputs.

Next, in search page, the tool will run either in "Search" mode or in "ShowAll" mode. This is done by specific SQL sentences according to user's input got from index page. It calls external JavaScript methods in "frontend.js" to realize the sorting algorithms. The sorting algorithms were referred and adapted from [104]. It can sort the result list according to numeral data such as id, year, score and recommendation (degree of relevance). The several Boolean parameters are initialized to switch the sorting "order" between ascend and descend. The "mode" attribute indicates which column's value to be used for sorting. The "mode" is set by indicating the column array's index using JavaScript DOM programming:

Listing 5.4: JavaScript Listing

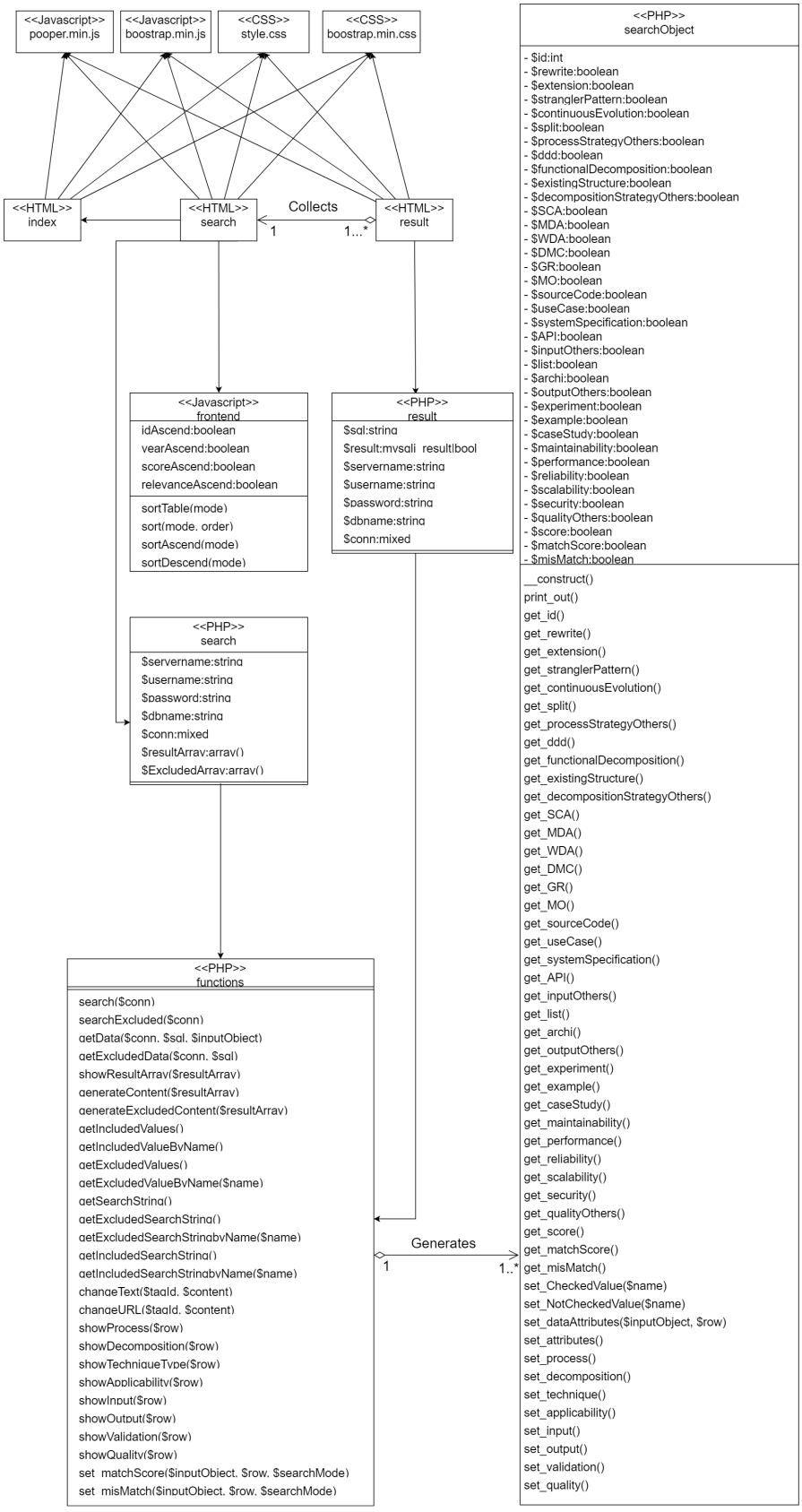


Figure 5.6: Class Diagram of the Web-based Tool

```
1 //Get the table element x by column index from HTML page
2 x = rows[i].getElementsByTagName("TD")[mode];
3 //Get the element content value y from x
4 y = Number(x.innerHTML);
```

According to each "y" value of the selected column, ascend or descend sorting is possible by switching the rows within the result list. However, if the sorting steps are casually executed according to the user's click, the result will be inconsistent, which brings confusion to the user, because the new sorting step will be executed based on the previous sorted list. For instance, if the user clicks "IDSort" first and then "YearSort", the result list order will be different from which if he/ she clicks "YearSort" first and then "IDSort". In order to enhance the reproducibility of sorting results, the order of sort steps was explicitly designed. Particularly, I fixed the sorting steps: first, the method would automatically sort the list in by ID in ascending order; then, the method would execute sorting according to the user's requirement; finally, the result was sorted according to higher recommendation score (relevance). In that case, it made sure that each time the most relevant approaches/ frameworks were visible at the top, and the result list kept consistent and reproducible.

The detailed source code is available at [104][40]. For better maintainability, the "sortTable" method applied multiple "sort" methods according to the user's input, and "sortAscend" as well as "sortDescend" methods were individual sorting methods developed for the future use.

Third, after the user clicks one title in the result list, the tool will open a new result page, showing all data of one contribution from the database.

Emphasis should be put on the shared classes and components. The HTML contents were defined in PHP file, and the code for database connection and access was also written in the PHP accordingly. According to Figure 5.6, "search.php" and "result.php" both contained parameters for database connection. In addition, they applied methods from "functions.php" to realize data retrial and content generation in HTML pages. Based on the parameters selected by the user and posted from index page, the SQL sentence (Listing 5.5) could be generated using the following pseudo equation:

Listing 5.5: SQL Pseudo Sentence Generated from User's Selection

```
$sqlForInclusion = "SELECT␣*␣FROM␣contribution␣WHERE␣(" . $includedParameters0 . "=1␣OR␣" .
    $includedParameters1 . "=1␣OR␣" . $includedParameters2 . "=1␣)␣AND␣(" .
    $excludeParameters0 . "=0␣AND␣" . $excludeParameters1 . "=0␣AND␣" . $excludeParameters2 .
    "=0␣)";
```

As we can see in the pseudo equation, all parameters selected as "Include" are assigned with Boolean value 1 and combined with "OR". Likewise, all parameters selected as "Exclude" are assigned with Boolean value 0 and combined with "AND". In that case, after SQL query, all matched contributions will not contain any parameters excluded by the user, and all possible contributions that contain at least one parameter desired by the user should be included in the result list. Besides, "Neutral" elements are ignored because the user also does not care about them.

What's more, "functions.php" initializes "searchObject" for each matched contribution in the result list and the user's selection in index page. These objects represented the parameters of the approaches or frameworks proposed by the contributions. By calling "setMatchScore" method in "functions.php", the tool will compare the user's input object with each candidate in the result list and generate the recommendation score (degree of relevance). Based on this equation, each candidate is assigned with a consistent score. And it will aid user to judge the feasibility of each proposed approach or framework. Specifically, if the user clicks "Search" button in index page, the Equation 5.1 is defined as follows:

$$RecommendationScore = [\frac{NumberOfSelectedParameters - NumberOfMissingParameters}{NumberOfSelectedParameters} * 5 + InitialQualityScore] * 10 \quad (5.1)$$

Please note that the initial quality score was defined as a five-point likert scale in the data extraction form, and it ranged from zero to five. So the output of Equation 5.1 is a numerical value that ranges between zero and one hundred. Additionally, if the user clicks "ShowAll" button in index page, the recommendation score will be calculated according to Equation 5.2, and its range keeps the same.

$$RecommendationScore = InitialQualityScore * 20 \tag{5.2}$$

Till now, I have briefly introduced techniques, tools, and implementation details about the web-based tool. After some debugging and testing on the local machine, the web-based tool was stable and worked as expected. All desired functionalities were realized successfully with considerably well time performance. Therefore, further steps could be taken to deploy the tool together with its database online.

5.6 Microsoft Azure Deployment

In this thesis, Microsoft Azure was applied to deploy the web-based tool online. According to the official introduction [71], it provides various products and services such as online virtual machine, cloud computation, IoT development and deployment, AI and machine learning, etc.

Microsoft Azure also provides Web App and MySQL database components, which supports PHP and online database. By means of these tools, the web-based tool can be accessed via a public URL from the external network. The detailed deployment steps were introduced in the README file of the web-based tool in GitHub [40], and the structure overview can be seen in Figure 5.7. Specifically, a Web App container and a corresponding database were created and configured in Azure. The local database was then imported into the online phpMyAdmin portal, whose basic operation was same as local phpMyAdmin as mentioned previously in Section 5.4. After the database was imported, the source code was migrated into the Web App container using KUDU environment. And it was further adapted in order to fit the online environment. The information about online database connection, such as server name, database name, user name, and password was retrieved from the online database. So the parameters of all source code files, which called methods for database connection, were changed according to this information. Specifically, in the web-based tool, "search.php" and "result.php" were changed accordingly.

Finally, after deployment was completed, users were able to visit this web-based tool externally by the URL provided by Microsoft Azure.

5.7 Tool Evaluation

The final step of development was to evaluate the tool. A good software involves not only good developers but also real users, who see and use the software from another perspective, in various ways and in different environments. They are more likely to encounter hidden bugs and errors, thus suggestions based on real-life scenarios can be obtained.

5.7.1 Target Test Participants

The targeted participants for evaluation were defined at first. Software architects, programmers, teachers and students of relevant majors were involved in testing and consultation. Besides, target users were supposed to have basic English ability.

5.7.2 Test Task and Questionnaire

Next, the desired test tasks and a questionnaire PDF were created to guide the users to conduct testing and gather feedback. The content of the questionnaire is available in Appendix B.

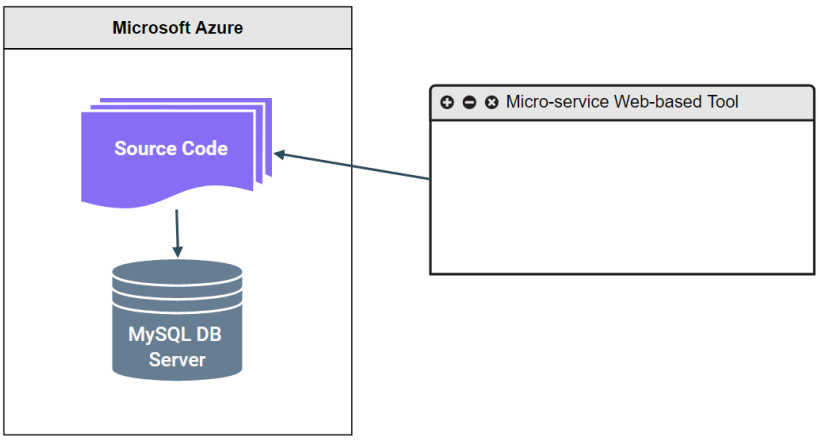


Figure 5.7: Structure of the Web-based Tool on Microsoft Azure

First, an introduction about the web-based tool was given in the questionnaire. Then some demographic data about test participants were collected for statistical purpose. Third, several small tasks together with a link to GitHub instructions were provided. The users could follow the tasks and use the tool to search for results. In addition, they could also customize their input and check whether this tool would generate suitable answers, and whether it was convenient and efficient. Finally, several questions were answered by users about their user experience and their judgment about the tool. Users could sign five-point likert score to each question and enter some advice in the text field.

Later, a proper invitation email together with the questionnaire PDF were sent to target participants. After they completed the testing, their questionnaire PDFs were sent back to me again for evaluation. Based on their practical working experience, valuable feedback about shortage, quality, effectiveness, and further improvements could be gathered.

5.7.3 Evaluation Result and Statistics

As a result, nine participants took part in the evaluation process. They read the instructions, tried the web-based tool, filled the questionnaires in, and sent them back. Then their advices and statistical data were gathered and visualized. Table 5.1 and Table 5.2 are the raw data gathered from user feedback.

The evaluation process involved two teachers, four students, two developers and one IT consultant. Since they studied different majors or had different working experience, they could provide a variety of feedback from their own perspective. In Table 5.1, The "Years of Professional Experience" and "Years of Experience with Microservices" represented their familiarity of this topic. The "Job" and "Major" indicated the participants' roles in their work and study.

In addition, the five-point likert scales in Table 5.2 shows the users' rating of the tool in four main aspects, say, understandability, usability, effectiveness, and meet of expectation. Understandability indicated whether users could understand the concept, terminology and functionalities provided by the tool; usability meant if the usage of the tool was convenient; effectiveness indicated if the tool could provide relevant and valuable information to the user during their architectural refactoring process; finally, it was also validated whether the tool could meet the users' requirement and expectation to reduce time and effort for decision-making and gathering information about microservice migration. Last but not least, the time duration of finishing the test was documented, this could also reflect the previous four accepts.

Table 5.1: Test Participants Demographic Data

No.	Job	Major	Years of Professional Experience	Years of Experience with Microservices
1	Researcher	/	5.0	0.5
2	Researcher	/	8.0	3.0
3	Student & Software Developer	Software Technique	1.0	0.5
4	IT Consultant	/	6.0	3.0
5	Software Developer	/	8.0	5.0
6	Student	IT	0.0	0.0
7	Student	IT	1.0	0.0
8	Student	IT	1.0	0.0
9	Software Developer	IT	2.0	2.0

Table 5.2: Test Participants Statistic Data

No.	Understandability	Usability	Effectiveness	Meet of Expectation	Time
1	3	2	2	2	10
2	3	2	3	2	20
3	2	3	4	4	5
4	4	4	2	3	15
5	3	5	4	3	5
6	5	4	3	3	30
7	4	4	2	5	15
8	4	4	4	3	10
9	3	2	4	4	5

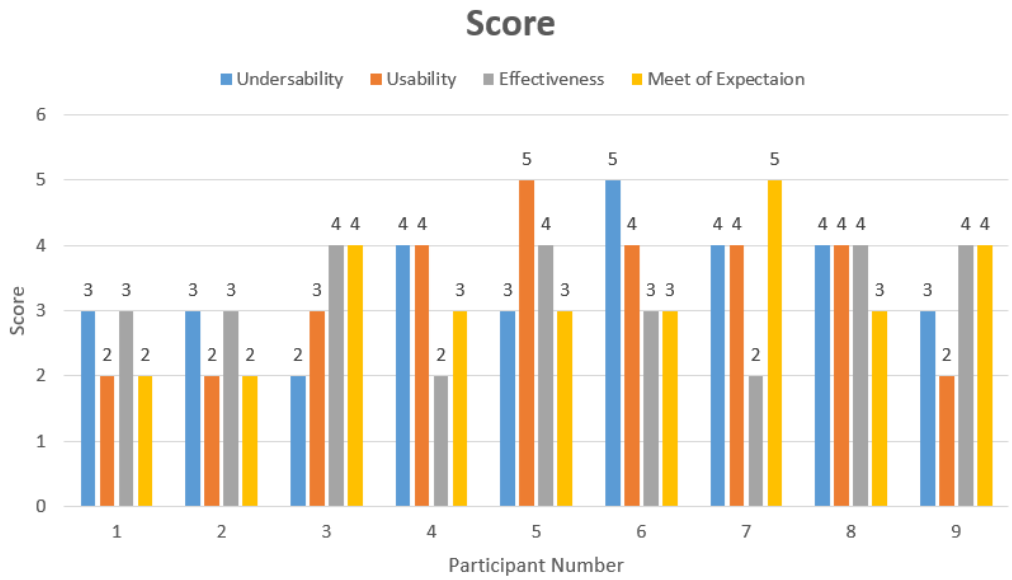


Figure 5.8: Statistic for Scores

The statistical analysis of data was then conducted in order to have an overview of users' feedback. According to Figure 5.8, 5.9, the distribution of each aspect is visualized. Additionally, the box plots 5.10, 5.11 are generated to inspect their mean, median, maximum, minimum and quartiles.

From the figures, we can see that the participants held different opinions on the web-based tool according to their own viewpoints. In detail, people who had more experience in working and programming tended to have more strict judgments. They had more requirements for this tool and expected it to have better usability and effectiveness. Besides, they needed more detailed explanation of each property proposed by the web-based tool. They also reported that the tool was sometimes unable to provide desired outputs according to their inputs. All these factors gave rise to the result that developers, IT consultants, and researchers rated the scores as relatively lower than those rated by students. Also, these experts reported that they spent less time than students to use the tool on average. This again proved that they were more familiar with the topic, and they could quickly understand the usage of the tool and its outputs.

When we refer to the box plots (Figure 5.10 and Figure5.11), we can see that the mean and median of the score were all higher than or equal to "3", which was half of the full point. The understandability showed the best variation among the four aspects because it had the highest mean and median. This meant that most participants agreed that the tool was pretty much understandable. They also reported that the tool usually could provide some information about migration suggestions. However, when it came to particular scenarios, it sometimes failed to show relevant results and could not meet their expectation. The last aspect was usability, according to user's feedback, it had the largest distribution, meaning that the user's had different opinions on it. Some people gave particular advice about the UI or functionalities to improve the usability, and they are shown in Table 5.3. Finally, the participants took about thirteen minutes on average to fulfill the test tasks, which was desired and expected as the result. It showed that most participants were able to use the tool after some minutes of trying and testing.

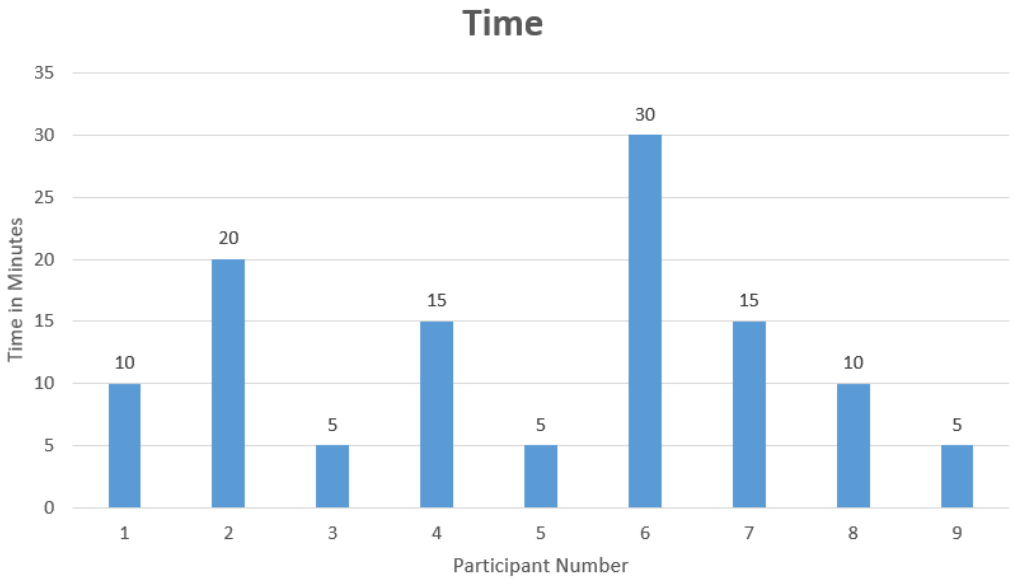


Figure 5.9: Statistic for Time

Figure 5.10: Score Variation

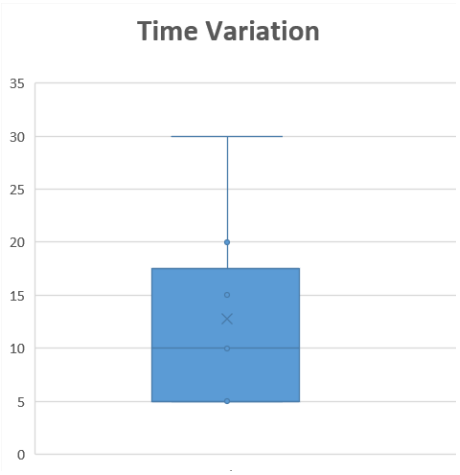


Figure 5.11: Time Variation

Table 5.3: User Feedback Advices

No.	Advice	Number of Mentions
1	Reset button is recommended	1
2	Clarify the result in introduction	1
3	"Include/neutral/exclude" radio buttons are not intuitive	2
4	Too many filtering parameters	1
5	Quality Metrics/ Intention part feels a little obsolete	1
6	Exlpain in detail the tool's functions and propose: how to help to make decisions about the migration	1
7	An explanation on each point/option would be helpful	2

Continued on next page

Table 5.3 – *Continued from previous page*

No.	Advice	Mentioned Times
8	The algorithm behind this filtering is unclear to user	1
9	Either place a "remove filters" button or always show a complete list of results when clicking "Search" but gray out those that are not relevant according to the selected search string	1
10	Is it necessary to differ between a "Score" and "Recommendation"?	1
11	The "Score" and its definition is unclear	1
12	"Recommendation" in % is not very helpful. A percentage-range mapping to a term would helpful (e.g. 0-20% = not worth, 20-40% = xyz etc)	1
13	The tool sometimes return no results after search	2
14	UI itself could be simplified by starting with generic selection and then getting more and more into detail, instead of having to select everything in the first place	1
15	Possibility of using NLP to automatically label new papers	1
16	Use Bootstrap to help to generate the front-end pages quickly	1

Besides demographic data, score data and time data, users' advice about further improvements were also collected in the questionnaire. I extracted the major advice and classified them into Table 5.3. Additionally, I counted the number of mentioned of each advice. This could show the most desired improvements proposed by the user, indicating a higher priority for change.

5.7.4 Suggested Improvements

According to Table 5.3, several improvements to the tool were made. First, As we can see in Figure 5.12, in index page, the "ShowAll" button was removed. Even though the "ShowAll" button also conducted searching in database, its functionality was similar to the "Search" button. This sometimes brought confusion to users. Therefore, the searching algorithm was adapted so that the "Search" button can also execute the method of "ShowAll", a simple conditioning logic was implemented to decide whether the user selected some parameters or not. If the user did not provide any input, the tool also automatically shows all data from the database.

Second, as requested by one participant, a "Reset" button replaced the "ShowAll" button so that users could clear their inputs anytime and refresh the index page to conduct a new search.

Third, many participants reported that the terminologies or elements in index page were not intuitive and required more description. Even if I implemented hover effect by Tooltips in Bootstrap, I later found that many users failed to identify it. In that case, I added question marks beside each section to notify the user that he/ she could check for hints about each section. Additionally, the introduction at the beginning and the instructions were supplemented by more detailed explanation.

Fourth, in search page, some participants said that the definition of "Score" and the difference between "Score" and "Recommendation" were vague. As mentioned before, "Score" was defined by me based on the quality, comprehensiveness and feasibility of contributions. This is an internal parameter for my research. In addition, "Recommendation" also included "Score" as one factor during calculation (Equation 5.1). So "Recommendation" already reflected "Score" to some extent. In order to eliminate confusion, the "Score" column and "ScoreSort" button were finally removed, which is shown in Figure 5.13.

Fifth, one developer suggested that the tool could show the gray out contributions that were irrelevant to the user's search results. I thought this was a good idea because the tool could

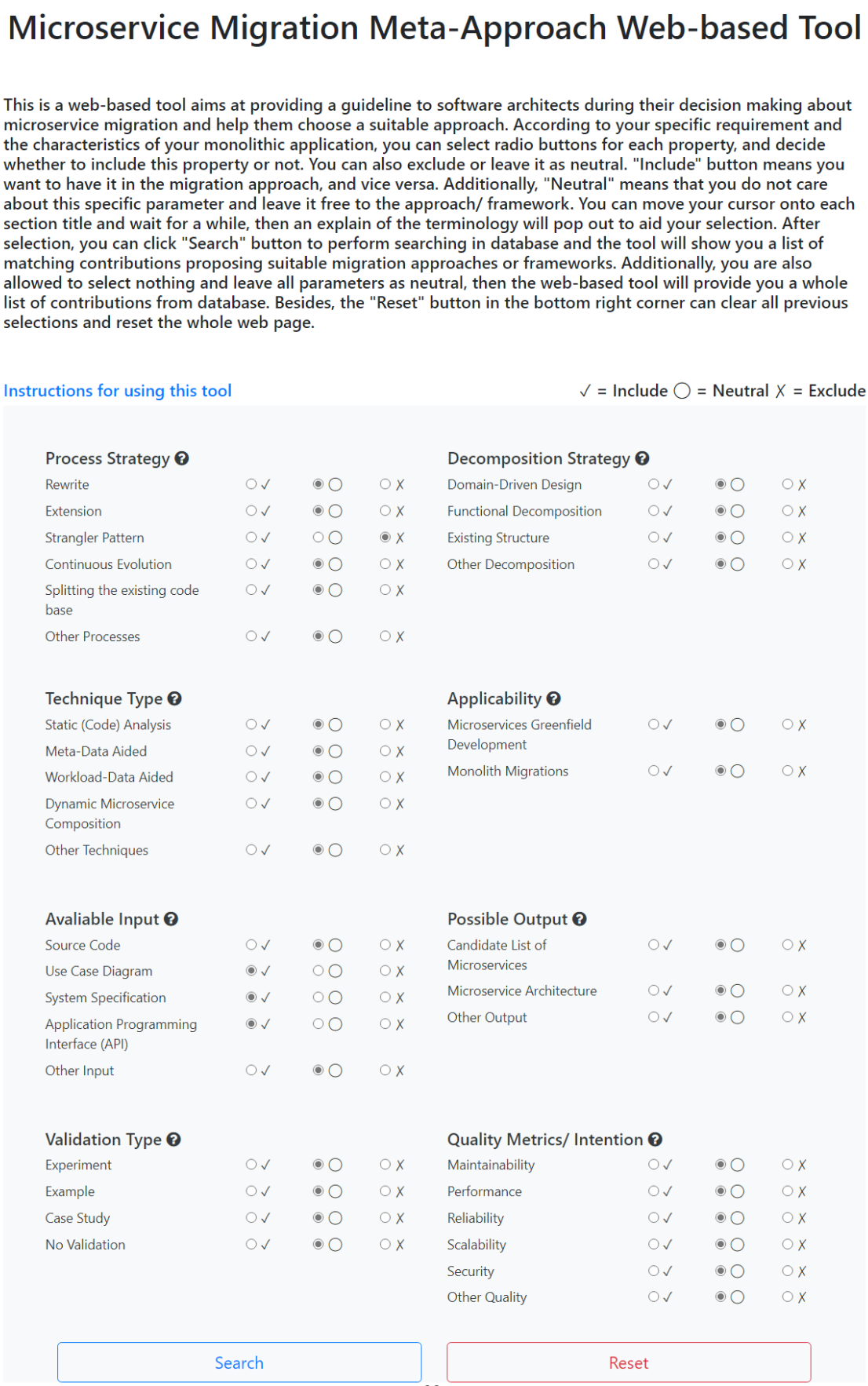


Figure 5.12: Improved Design of the Index Page

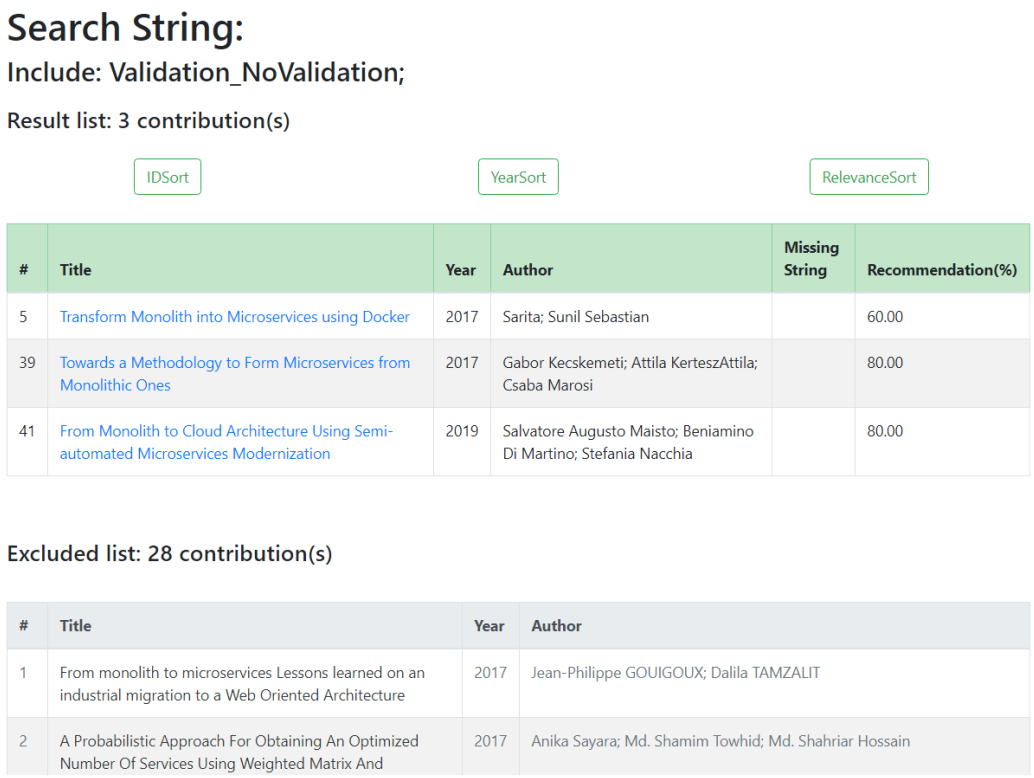


Figure 5.13: Improved Design of the Search Page

always provide users a comprehensive result about which contributions were included or excluded. To realize this, another SQL sentence was designed to get the excluded contributions. Please note that the parameter "sqlForInclusion" was defined previously in Listing 5.5 to get included contributions.

Listing 5.6: SQL Pseudo Sentence Generated to Get Excluded Contributions

```
$sqlForExclusion = "SELECT␣*␣FROM␣contribution␣WHERE␣id␣NOT␣IN␣(" . $sqlForInclusion . ")";
```

After SQL query, the excluded contributions will compose a gray out list. It is shown under the included list. The number of excluded contributions is also visible to users.

Finally, the order of the sections in result page was also adjusted so that it was logically more fluent to users.

Some other advice were also valuable, but I decided not to change it because of the original design purpose. For instance, one participant said that the "Quality Metrics/ Intention" part was obsolete. Indeed, this study referred to the ISO25010 [3]. I thought it could already involve main qualities and intentions considered in microservice migration scenarios because the metrics discussed in most contributions were similar to those in ISO25010. For further research, it is recommended to study more other aspects. Other advice such as No.4 and No.14 both said that the parameters were complex and too many. In this research, due to time reason, the tool aimed at providing simple and clear functionalities and UI to serve for architectural refactoring. However, it is worthwhile if the filtering steps and the UI can be re-structured in the future. Also, No.8 reported that the filtering algorithm was unclear to the user. Considering the fact that the "Include" string, "Exclude" string, and "Missing String" column in search page could already provide enough information about filtering to users, I decided to hide the SQL sentence to users in order to avoid confusion.

Because of time and effort limitation, six advice were taken (No.1, 3, 6, 9, 20, 22) and changes

on the tool were conducted. Remaining advice were also valuable for further improvements, such as much more detailed explanation for the result list and parameters in each section; automatic labeling for new papers by NLP; quick front-end pages realization by Bootstrap; "Recommandation" percentage to term mapping; and inclusion of more contributions to avoid empty search result.

In the end, the improvement of the web-based tool made it easier to use and more understandable. According to users' feedback, they thought the development of such tool was a creative idea to assist the architectural refactoring. It was of great potential to be further developed and improved to be more comprehensive and complete.

Chapter 6

Discussion

In this section, I will interpret the results generated from the systematic literature review and web-based tool implementation as well as the feedback gathered from questionnaires.

In the systematic literature review, three main groups of data for the migration process were extracted: paper context data, empirical data, and quality assessment. According to the previous studies and thirty-one contributions, the empirical data and quality assessment were further partitioned into eight main sections: process strategy, decomposition strategy, technique type, applicability, required inputs, expected outputs, validation type and intentions/ quality metrics.

Regarding process strategy, splitting the existing code base, continuous evolution became popular, unlike the previous research by Fritzsch et al. [34] where rewrite and strangler pattern were commonly applied during migration. This fact is supported in Figure 4.6, it is obvious that these two strategies were applied most frequently by the development teams. One reason for this significant shift may be the migration costs of rewrite and strangler pattern tend to be higher in the industry because both of them require rewrite and redeployment of the existing parts to some extent, which is often costly in terms of time, effort, and required resources.

In my study, I also noticed that the number of monolith migrations (brownfield development) was higher than microservice greenfield developments. This fact was consistent with the previous research [33]. An assumption can be made that when the companies meet technical restrictions or maintenance problems with the monolithic applications after several years of operation, the willingness to adopt a new platform or architecture becomes stronger. Besides, since they have the basis of legacy resources such as source codes, databases, system specifications, etc., it becomes easier for them to refactor the application or modernize it using new technologies other than implementing a completely new application from the beginning.

The technique types classified from my literature review mainly referred to the definition by Fritzsch et al. [33], say, Static Code Analysis Aided (SCA) method, Meta-Data Aided (MDA) method, Workload-Data Aided (WDA) method and Dynamic Microservice Composition (DMC) method. Specifically, SCA played a dominant role in application analysis, following with the WDA. Additionally, DMC and MDA usually acted as supplementary techniques to inspect the system in different perspectives. This finding supports the result given by Ponce et al. [84], but it is different from the overview conducted by Fritzsch et al. [33]. Indeed, the applied techniques varied differently according to specific scenarios. Each approach will choose its own techniques which fit its requirement and condition most. However, considering the fact that Ponce and his colleagues [84] reviewed twenty contributions and I have reviewed thirty-one contributions, while Fritzsch et al. [33] included ten contributions in total during their literature review. I assumed my results are much general under various situations.

The required inputs and expected outputs sections in this study further extended the research by Fritzsch et al. [33]. As I have discussed, source code was the most common inputs for the migration approach, and it was often combined with other resources. On the other hand, a candidate list of microservices and a suggested architectural design of microservice application were usually generated after the execution of proposed approaches. Additionally, new tool support such as

ENTICE or Docker [51][91] were identified which enable automatic deployment. Besides, other output results such as partition diagram [87][108][100][42] also corroborated the study by Fritzsch et al [33].

The validation types documented in the included contributions supported the statistic result by Ponce et al. [84] as well. Experiments and case studies were frequently introduced in purposed approaches or frameworks.

Finally, my systematic literature review investigated the intentions and evaluated quality metrics mentioned among the contributions. These metrics referred to ISO25010 [44]. Same as previous studies [34] [84] [99] [98] [17], maintainability and scalability were the two most essential metrics which drove the development teams to migrate their monolithic applications. Other metrics, such as performance, reliability, flexibility, etc., were also listed in previous research [34][99][98]. This provided a deeper overview of the drivers and desired qualities expected by users during their migration process, which is worthy of being referred by new microservice migration projects in the future.

Based on the knowledge acquired from my literature review, I have developed a web-based tool to assist the migration approach selection. The feedback provided by test participants during the evaluation stage showed that the scores rated by participants in each section were all higher than or equal to "3" in five-point likert scale (the five-point likert scale ranges from zero to five, zero means fail, while five means excellent), indicating that the tool was able to support the users to fulfill this objective. However, this tool still required improvements in terms of usability and repository extension of available contributions.

Chapter 7

Threats to Validity

In this section, the potential threats to the validity of this study will be mentioned.

(1) During the systematic literature review, informal review process and vague definition of scope will make the result inconsistent or not comprehensive. So my literature review followed an explicit guideline which was recommended by Kitchenham and Charters [52]. Therefore, the repeatability as well as reproducibility of the results could be ensured [33].

(2) It is possible that some important contributions or results are not captured in the field of study. The main reason for this is because the research scope is limited or insufficient contributions are retrieved. To mitigate this threat, four online databases were used for contribution searching and selecting. Therefore, adequate amount of primary studies were reviewed and selected. Due to time reason, snowballing was not extensively performed. But the contributions within the repository should be able to provide a comprehensive overview of this topic.

(3) Potential risk of research bias exists in every literature review always [52]. The data gathered by researchers usually depend on their own understanding of the results, so that the research results may vary among different researchers. In my study, the contribution inclusion decisions were made by two researchers, so that the review bias can be mitigated to some extent. For future research, it is commanded to involve more researchers to minimize the bias further.

(4) Potential threat to data correctness might exist in my study. To overcome this, I designed a data extraction form to ensure data correctness and consistency. But the data extraction and data synthesis was done by only one researcher. In the future, more researchers should work together during the data extraction and synthesis process to eliminate this threat.

(5) The proposed migration framework and corresponding web-based tool were developed, and they were evaluated by nine participants involving software architects, programmers, teachers and students of relevant majors. Retrieving the real and complete user feedback is always a challenge. Because the test participants sometimes have no idea about the tool's objectives and functionalities, they may also feel difficult to express their experience. Therefore, a formal evaluation procedure, including user instructions, test tasks and questionnaires, was designed explicitly. Based on their different background, valuable feedback could be acquired from them. Negative comments were also encouraged by guaranteeing confidentiality and anonymity during the evaluation procedure [34]. However, the evaluation process was limited within academic area. For further research, it is suggested to be applied in the real industry environment and tested with real projects.

Chapter 8

Conclusion

Microservice migration is a popular topic in the industry, but an overview of appropriate migration approaches or frameworks is missing. This brings challenges to software architects and developers during their decision-making for architectural refactoring, especially when they face with complex legacy applications.

In this paper, I conducted a systematic literature review about architecture refactoring from monolithic applications to microservices. After defining an explicit research protocol, thirty-one primary contributions with empirical data were selected, studied, and corresponding migration information was gathered using data extraction form and stored in a repository. These processes were conducted by two researchers so that research bias was minimized. Besides, a corresponding web-based tool was developed and tested by several consulting experts and students.

Based on previous researches [33][34][84][98][99][17], in my study, I classified the identified approaches or frameworks according to eight main aspects: required inputs, expected output, technique type, decomposition strategy, process strategy, applicability, validation type, tool support and intentions or quality metrics concerned during migration. For each aspect, I further defined several parameters which were common practices in industry, and statistic information about the adopted times of these parameters in the proposed approaches or frameworks was also calculated. In that case, readers could have a brief overview of various migration practices.

Based on my systematic literature review, a migration framework focusing on technique and process was designed. It took the above mentioned eight main aspects as inputs to suggest migration solutions. And then, a web-based tool was developed using HTML, JavaScript, PHP, Bootstrap, MySQL and XAMPP accordingly. It stored all extracted migration information into a MySQL database. It served to help software architects and developers choose suitable refactoring techniques and approaches according to their expectations and application environments. In order to evaluate the effectiveness of the tool, it was deployed using Microsoft Azure and could be visited from external networks using a unique URL address. Several test tasks and corresponding feedback questionnaire were designed and sent to nine test participants, including software architects, programmers, teachers and students of relevant majors for testing. The feedback given by the participants showed that the tool was able to realize anticipated objectives. But a considerable amount of advice and the evaluation scores indicated that the tool still had great potential to be improved in terms of usability, understandability, effectiveness, and the number of included approaches/ frameworks in the repository.

Additionally, my study also focuses on the intentions or quality metrics concerned during migration. Because they reflect the weakness of existing application or the characteristics of new microservices which development teams want to achieve. They also indicate the causes and effects before and after the architectural refactoring. In my study, I referred to ISO25010 [44] and previous studies [34] [84] [99] [98] [17] to address five main quality attributes: maintainability, performance, reliability, scalability, and security. They were the frequently inspected attributes in these studies. Besides, other metrics including cost, team communication, complexity, etc. were also documented in my repository as other remaining aspects. My study provided an overview of the drivers and

desired qualities expected by users during architectural refactoring, but I only limited my scope within ISO25010 standard [44], which might fail to meet current research direction or users' requirements. So it is recommended to conduct deeper research about more intentions and quality metrics in the future.

Some problems also appeared in this study. The vague and mixed definition of framework and approach often provided confusion during contribution reading and classification. My study also revealed that microservice migration approaches or frameworks still varied according to different scenarios and environments. Besides, formal migration frameworks proposed by researchers accounted for only 10% of the total amount. In addition, the quality of each contribution also varied greatly because of the authors' experience. What's more, the tool should not only be evaluated in academic area but also tested by practice, so that the evaluation result can be better ensured. Finally, some participants claimed that the quality aspects defined in this study were obsolete. They could not reflect migration intentions and quality of microservices nowadays.

For further study, I suggest including more researchers during systematic literature review to conduct thesis selection and data extraction. Searching in more online databases and performing snowballing research will also help to collect relevant contributions further and extend the migration information repository. In addition, contributions written in other languages, especially German, can also be investigated to broaden the review scope and retrieve a more general result. Also, more researchers should be involved in conducting the systematic literature review to eliminate research bias and enhance the correctness of the review result. The intentions and quality attributes can be further investigated by referring to other definitions or defining new metrics in addition to ISO25010 [44]. Moreover, the quality of the contribution should be evaluated in a much more formal way using an explicit and quantifiable equation. Finally, further improvement of the web-based tool according to the remaining expert's advice in Table 5.3 is also recommended.

Additional scopes of research can be: (1) performance comparison of different migration approaches or frameworks based on the same legacy application, including consumed times, costs, required workers, etc.; (2) finding a solution which can identify new possible contributions about microservice migration from the online database automatically and periodically, possibly identifying them according to the content of title and abstract, by means of machine learning technology and word-based analysis algorithms; (3) deeper investigation in quality metrics and intentions is also suggested, and new attributes should be defined and compared among different approaches, for instance, portability, compatibility, better team organizations, and so on. Portability and compatibility play an essential role nowadays because of the popularization of mobile devices and IoT devices. It is curious how microservice applications can support these platforms. Since software development always involves human factors, the influence of team organization on microservice migration should also be investigated.

Appendix A

Contribution Index

This is the contribution index which contains primary contributions after first-time abstract reading. General information such as title, source database, include, exclude decision and notes is provided.

Table A.1: Contribution Index

No.	Title	Database	Selection	Note
1	From monolith to microservices: Lessons learned on an industrial migration to a Web Oriented Architecture [38]	IEEE	Included	Full text read
2	A Probabilistic Approach For Obtaining An Optimized Number Of Services Using Weighted Matrix And Multidimensional Scaling [92]	IEEE	Included	Full text read
3	Microservices architecture: Case on the migration of reservation-based parking system [108]	IEEE	Included	Full text read
4	Functionality-oriented Microservice Extraction Based on Execution Trace Clustering [47]	IEEE	Included	Full text read
5	Transform Monolith into Microservices using Docker [91]	IEEE	Included	Full text read
6	TheArchitect: A Serverless-Microservices Based High-level Architecture Generation Tool [80]	IEEE	Excluded	Only discussed tool support for development but not migration
7	Visualization Tool for Designing Microservices with the Monolith-first Approach [74]	IEEE	Included	Full text read
8	A Rule-based System for Automated Generation of Serverless-Microservices Architecture [79]	IEEE	Excluded	Only discussed development using tools from No. 6 but not migration
9	Microservices: Migration of a Mission Critical System [68]	IEEE	Included	Full text read

Continued on next page

Table A.1 – *Continued from previous page*

No.	Title	Database	Selection	Note
10	"Functional-first" recommendations for beneficial microservices migration and integration: Lessons Learned from an Industrial Experience [39]	IEEE	Excluded	Not detailed enough as an empirical report
11	Extracting Candidates of Microservices from Monolithic Application Code [49]	IEEE	Included	Full text read
12	Migrating Legacy Software to Microservices Architecture [50]	IEEE	Included	Full text read
13	Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications [23]	IEEE	Included	Full text read, literature review as reference
14	Automatic performance monitoring and regression testing during the transition from monolith to microservices [46]	IEEE	Included	Full text read
15	From Monolithic Architecture to Microservices Architecture [26]	IEEE	Excluded	Solution proposal
16	From Monolith to Microservices: A Dataflow-Driven Approach [20]	IEEE	Excluded	Already covered in previous study[83]
17	Making the Move to Microservice Architecture [56]	IEEE	Excluded	Technique irrelevant topic
18	A Framework for Evaluating Continuous Microservice Delivery Strategies [58]	ACM	Excluded	Irrelevant about migration
19	MAGMA: Build Management-based Generation of Microservice Infrastructures [106]	ACM	Excluded	Only discussed tool support for development but not migration
20	Research on Digital Publishing Application System Based on Micro-Service Architecture [10]	ACM	Excluded	Irrelevant about migration
21	An Efficient Algorithm of Context-Clustered Microservice Discovery [62]	ACM	Excluded	Irrelevant about migration
22	Microservice Architecture and Model-driven Development: Yet Singles, Soon Married (?) [86]	ACM	Excluded	Irrelevant about migration
23	Microservice Architecture in Industrial Software Delivery on Edge Devices [59]	ACM	Excluded	Irrelevant about migration
24	Towards Defining a Microservice Migration Framework [7]	ACM	Excluded	Only conceptual framework definition methodology without actual experience was proposed
25	Tracking and Controlling Microservice Dependencies [36]	ACM	Excluded	Irrelevant about migration
26	Migrating Web Applications from Monolithic Structure to Microservices Architecture [87]	ACM	Included	Full text read

Continued on next page

Table A.1 – *Continued from previous page*

No.	Title	Database	Selection	Note
27	A Model-driven Workflow for Distributed Microservice Development [85]	ACM	Excluded	Irrelevant about migration
28	Research on Optimization of Course Selection System Based on Micro service and Dynamic Resource Extension [61]	ACM	Excluded	Irrelevant about migration
29	A Microservice Architecture for Online Mobile App Optimization [111]	ACM	Excluded	Irrelevant about migration
30	The Applicability of Palladio for Assessing the Quality of Cloud-based Microservice Architectures [53]	ACM	Excluded	Irrelevant about migration
31	A logical architecture design method for microservices architectures [90]	ACM	Included	Full text read
32	Availability and Scalability Optimized Microservice Discovery from Enterprise Systems [25]	Springer	Included	Full text read
33	Microservices Identification Through Interface Analysis [9]	Springer	Excluded	Already covered in previous study[33]
34	A Model-Driven Approach Towards Automatic Migration to Microservices [13]	Springer	Included	Full text read
35	Supporting the Decision of Migrating to Microservices Through Multi-layer Fuzzy Cognitive Maps [21]	Springer	Included	Full text read, literature review as reference
36	Evaluation of Microservice Architectures: A Metric and Tool-Based Approach [29]	Springer	Excluded	Irrelevant about migration
37	Migration to Microservices: Barriers and Solutions [37]	Springer	Included	Full text read, literature review as reference
38	Challenges When Moving from Monolith to Microservice Architecture [48]	Springer	Excluded	Literature review and irrelevant about migration
39	Towards a Methodology to Form Microservices from Monolithic Ones [51]	Springer	Included	Full text read
40	Translating a Legacy Stack to Microservices Using a Modernization Facade with Performance Optimization for Container Deployments [64]	Springer	Included	Full text read
41	From Monolith to Cloud Architecture Using Semi-automated Microservices Modernization [65]	Springer	Included	Full text read
42	From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [76]	Springer	Included	Full text read
43	Tool Support for the Migration to Microservice Architecture: An Industrial Case Study [82]	Springer	Included	Full text read
44	Re-architecting OO Software into Microservices A Quality-Centred Approach [93]	Springer	Included	Full text read

Continued on next page

Table A.1 – *Continued from previous page*

No.	Title	Database	Selection	Note
45	An Experience Report from the Migration of Legacy Software Systems to Microservice Based Architecture [24]	Springer	Included	Full text read
46	Strategies Reported in the Literature to Migrate to Microservices Based Architecture [95]	Springer	Included	Full text read, literature review as reference
47	A Reconfigurable Microservice-Based Migration Technique for IoT Systems [97]	Springer	Included	Full text read
48	Identifying Microservices Using Functional Decomposition [103]	Springer	Included	Full text read
49	Microservices: Migration of a Mission Critical System [28]	Google-Arxiv	Excluded	Same as No. 9
50	Microservice Decomposition via Static and Dynamic Analysis of the Monolith [57]	Springer	Included	Full text read
51	Microservices migration patterns [8]	Google-Willy	Excluded	Various conceptual patterns for reference
52	An adaptive plan-oriented and continuous software migration to cloud in dynamic enterprises [73]	Google-Willy	Excluded	Could migration rather than microservice migration
53	A Design with Mobile Agent Architecture for Refactoring A Monolithic Service into Microservices [42]	Google-Others	Included	Full text read
54	A Model-Driven Approach to Microservice Software Architecture Establishment [101]	Google-Others	Excluded	Irrelevant about migration
55	From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining [100]	Google-Others	Included	Full text read
56	A New Decomposition Method for Designing Microservices [5]	Google-Others	Included	Full text read
57	Migration of Software Components to Microservices: Matching and Synthesis [22]	Google-Others	Included	Full text read
58	Use Case Driven Microservices Architecture Design [66]	Google-Others	Included	Full text read
59	Decomposition of monolithic web application to microservices [110]	Google-Others	Excluded	Bachelor thesis
60	Using Microservices for Legacy Software Modernization [54]	IEEE	Included	Snowballing
61	Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report [31]	IEEE	Excluded	Snowballing, already covered in previous study[83]

Appendix B

Evaluation Questionnaire

The following PDF file is the questionnaire designed for web-based tool evaluation. It contains an introduction about the web-based tool, usage instructions, test tasks and feedback form. In the questionnaire, some demographic data about test participants were collected for statistical purpose. The users could follow the instructions and conduct test tasks to search for results. Several questions were asked about user experience and their satisfaction. Users could sign five-point likert score to each question and enter some advice in the text field. After they completed the testing, their questionnaire PDFs were sent back for evaluation.

Microservice Migration Web-based Tool

Validation Questionnaire

This questionnaire will ask you several questions about your experience of using the microservice web-based tool. This tool is a part of the topic: *"A Meta-Approach to guide Architectural Refactoring from Monolithic Applications to Microservices."*, which is the Master Thesis by Qiwen Gu, a student of INFOTECH, University of Stuttgart. The aim of this tool is to provide a guideline to software architects during their decision making about microservice migration and help them choose a suitable approach.

First, some basic demographic data is needed for statistic purpose.

User Information:

Current job / role:	☐ Software developer	☐ Software architect	☐ Others:
	☐ Teacher	☐ Student	
Course of study (if student):	☐ Compu science	☐ Information technology	☐ Others:
years of professional experience:	☐	Year(s)	
Years of experience with microservices:	☐	Year(s)	

☐ **I admit that my information can be used as anonymized data in the publication.**

Next, the user's manual is available in GitHub in Section *"Running the web-based tool"*. The following section provides some instructions for the validation of this tool. You can use the tool by clicking this link.

1. In ***index page***, you can click the *"ShowAll"* button and get a complete list of contributions from database in ***search result page***.
2. In ***index page***, you can select some specific requirement and the characteristics of your monolithic application. For instance, Include: Process Strategy – Rewrite, Decomposition

Strategy - Functional Decomposition; Exclude: Technique Type - Meta-Data Aided; Other characteristics are left as Neutral. Then you can click the *"Search"* button and get a matching list of contributions from database in ***search result page*** (9 contributions).

3. In ***search result page***, you can click sort buttons and sort the result list in ascend or decently order by *"ID", "Year of publication", "Score" or "Recommendation"*. *"Score"* indicates the quality of the contribution, while *"Recommendation"* indicates the matching level of the contribution according to user's input.
4. If you click one contribution title in ***search result page***, you can read all detailed information gathered from it in ***detail page***, including: Programming language, Process strategy, Decomposition Strategy, etc. The result should provide you an overview about a recommended approach about how to migrate a specific monolithic application to microservices.

You can try this tool and customize your own input depending on your need, and check whether this tool can provide you with suitable answers, and whether this process is convenient and the application logic is understandable to you.

After using this tool, please fill in this form and provide me with your valuable feedback.

Questions: *Number indicates the score, 1 is lowest, 5 is highest*

	1	2	3	4	5
1. How do you think about the understandability of the user interface?	○	○	◉	○	○
2. How is the usability of this tool?	○	○	◉	○	○
3. To what extent do you think the tool is helpful to your work?	○	○	◉	○	○
4. Did the result meet your expectation?	○	○	◉	○	○

5. How long did you take to get the final result? ☐ Minute(s)

6. Do you have any advice for improvement regarding this tool?

Thank you very much for your time to participate in this survey to validation the result of this Master Thesis! I appreciate your precious and valuable feedback and this will allow further improvements!

You could send this form back via Email.

Thank you again for your cooperation!

Bibliography

[1] Marathon - a container orchestration platform for mesos and dc/os. https://mesosphere.github.io/marathon/.

[2] Production-grade container orchestration. http://kubernetes.io/.

[3] Iso / iec 25010 : 2011 systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. 2013.

[4] Swarm mode overview, Oct 2020. https://docs.docker.com/engine/swarm/.

[5] Omar Al-Debagy and Peter Martinek. A new decomposition method for designing microservices. *Periodica Polytechnica Electrical Engineering and Computer Science*, 63(4):274–281, 2019. https://pp.bme.hu/eecs/article/view/13925.

[6] Friends Apache. Xampp installers and downloads for apache friends. https://www.apachefriends.org/index.html.

[7] Florian Auer, Michael Felderer, and Valentina Lenarduzzi. Towards defining a microservice migration framework. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3234152.3234197.

[8] Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, and Theo Lynn. Microservices migration patterns. *Software: Practice and Experience*, 48(11):2019–2042, 2018. https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2608.

[9] Luciano Baresi, Martin Garriga, and Alan De Renzis. Microservices identification through interface analysis. In Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen, editors, *Service-Oriented and Cloud Computing*, pages 19–33, Cham, 2017. Springer International Publishing.

[10] Wang Bin, Yang Shulin, Ren Xuelei, and Wang Guyang. Research on digital publishing application system based on micro-service architecture. In *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, ICNCC 2017, page 140–144, New York, NY, USA, 2017. Association for Computing Machinery. https://doi.org/10.1145/3171592.3171613.

[11] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research 3*, page 993–1022, Mar 2003. http://www.cse.cuhk.edu.hk/irwin.king/_media/presentations/latent_dirichlet_allocation.pdf.

[12] Tooltips Bootstrap. Tooltips. https://getbootstrap.com/docs/4.0/components/tooltips/.

[13] Antonio Bucchiarone, Kemal Soysal, and Claudio Guidi. A model-driven approach towards automatic migration to microservices. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 15–36, Cham, 2020. Springer International Publishing.

[14] Alex Buck and Marc Wilson. Strangler pattern - cloud design patterns, Jun 2017. https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler.

[15] Jordi CabotI. Comparing domain-driven design with model-driven engineering, Sep 2017. https://modeling-languages.com/comparing-domain-driven-design-model-driven-engineering/.

[16] Ivan Candela, Gabriele Bavota, Barbara Russo, and Rocco Oliveto. Using cohesion and coupling for software remodularization: Is it enough? *ACM Trans. Softw. Eng. Methodol.*, 25(3), June 2016. https://doi.org/10.1145/2928268.

[17] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello, and Maria Julia de Lima. Analysis of the criteria adopted in industry to extract microservices. In *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*, CESSER-IP '19, page 22–29. IEEE Press, 2019. https://doi.org/10.1109/CESSER-IP.2019.00012.

[18] Ding-Kai Chen, Hong-Men Su, and Pen-Chung Yew. The impact of synchronization and granularity on parallel systems. volume 18[2SI], page 239–248, 01 1990.

[19] L. Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54, 2015.

[20] R. Chen, S. Li, and Z. Li. From monolith to microservices: A dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 466–475, 2017.

[21] Andreas Christoforou, Martin Garriga, Andreas S. Andreou, and Luciano Baresi. Supporting the decision of migrating to microservices through multi-layer fuzzy cognitive maps. In Michael Maximilien, Antonio Vallecillo, Jianmin Wang, and Marc Oriol, editors, *Service-Oriented Computing*, pages 471–480, Cham, 2017. Springer International Publishing.

[22] Andreas Christoforou, Lambros Odysseos, and Andreas S. Andreou. Migration of software components to microservices: Matching and synthesis. In *ENASE*, 2019.

[23] M. Cojocaru, A. Uta, and A. Oprescu. Attributes assessing the quality of microservices automatically decomposed from monolithic applications. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 84–93, 2019.

[24] Hugo Henrique S. da Silva, Glauco de F. Carneiro, and Miguel P. Monteiro. An experience report from the migration of legacy software systems to microservice based architecture. In Shahram Latifi, editor, *16th International Conference on Information Technology-New Generations (ITNG 2019)*, pages 183–189, Cham, 2019. Springer International Publishing.

[25] Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Colin Fidge, and Artem Polyvyanyy. Availability and scalability optimized microservice discovery from enterprise systems. In Hervé Panetto, Christophe Debruyne, Martin Hepp, Dave Lewis, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems: OTM 2019 Conferences*, pages 496–514, Cham, 2019. Springer International Publishing.

[26] L. De Lauretis. From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 93–96, 2019.

[27] Docker. Empowering app development for developers, Sep 2020. https://www.docker.com/.

[28] Nicola Dragoni, Schahram Dustdar, Stephan T. Larsen, and Manuel Mazzara. Microservices: Migration of a mission critical system, 2017.

[29] Thomas Engel, Melanie Langermeier, Bernhard Bauer, and Alexander Hofmann. Evaluation of microservice architectures: A metric and tool-based approach. In Jan Mendling and Haralambos Mouratidis, editors, *Information Systems in the Big Data Era*, pages 74–89, Cham, 2018. Springer International Publishing.

[30] E. Evans and M. Fowler. *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004. https://books.google.de/books?id=7dlaMs0SECsC.

[31] C. Fan and S. Ma. Migrating monolithic mobile application to microservice architecture: An experiment report. In *2017 IEEE International Conference on AI Mobile Services (AIMS)*, pages 109–112, 2017.

[32] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007. https://science.sciencemag.org/content/315/5814/972.

[33] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann. From monolith to microservices: A classification of refactoring approaches. *Lecture Notes in Computer Science*, page 128–141, 2019. http://dx.doi.org/10.1007/978-3-030-06019-0_10.

[34] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann. Microservices migration in industry: Intentions, strategies, and challenges. 10 2019.

[35] Andrew Gemino and D. Parker. Use case diagrams in support of use case modeling: Deriving understanding from the picture. *J. Database Manag.*, 20:1–24, 2009.

[36] Silvia Esparrachiari Ghirotti, Tanya Reilly, and Ashleigh Rentz. Tracking and controlling microservice dependencies. *Commun. ACM*, 61(11):98–104, October 2018. https://doi.org/10.1145/3267118.

[37] Javad Ghofrani and Arezoo Bozorgmehr. Migration to microservices: Barriers and solutions. In Hector Florez, Marcelo Leon, Jose Maria Diaz-Nafria, and Simone Belli, editors, *Applied Informatics*, pages 269–281, Cham, 2019. Springer International Publishing.

[38] J. Gouigoux and D. Tamzalit. From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 62–65, 2017.

[39] J. GOUIGOUX and D. TAMZALIT. "functional-first" recommendations for beneficial microservices migration and integration lessons learned from an industrial experience. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 182–186, 2019.

[40] Qiwen Gu. Tschiwengu/masterthesis, Aug 2020. https://github.com/TschiwenGu/MasterThesis.

[41] Matt Heusser. Refactor vs. rewrite: Deciding what to do with problem software, May 2020. https://searchapparchitecture.techtarget.com/tip/Refactor-vs-rewrite-Deciding-what-to-do-with-problem-software.

[42] Masayuki Higashino, Toshiya Kawato, and Takao Kawamura. A design with mobile agent architecture for refactoring a monolithic service into microservices. 02 2018.

[43] IEEE. About ieee. https://www.ieee.org/about/index.html.

[44] ISO/IEC 25010. ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models, 2011. https://www.bibsonomy.org/bibtex/25951b0998b7eaea346d826fd77110a48/bcoldewey.

[45] Jagadeesh Jagarlamudi, Hal Daumé, and Raghavendra Udupa. Incorporating lexical priors into topic models. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, page 204–213, USA, 2012. Association for Computational Linguistics.

[46] A. Janes and B. Russo. Automatic performance monitoring and regression testing during the transition from monolith to microservices. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 163–168, 2019.

[47] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai. Functionality-oriented microservice extraction based on execution trace clustering. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 211–218, 2018.

[48] Miika Kalske, Niko Mäkitalo, and Tommi Mikkonen. Challenges when moving from monolith to microservice architecture. In Irene Garrigós and Manuel Wimmer, editors, *Current Trends in Web Engineering*, pages 32–47, Cham, 2018. Springer International Publishing.

[49] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo. Extracting candidates of microservices from monolithic application code. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 571–580, 2018.

[50] J. Kazanavičius and D. Mažeika. Migrating legacy software to microservices architecture. In *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–5, 2019.

[51] Gabor Kecskemeti, Attila Kertesz, and Attila Csaba Marosi. Towards a methodology to form microservices from monolithic ones. In Frédéric Desprez, Pierre-François Dutot, Christos Kaklamanis, Loris Marchal, Korbinian Molitorisz, Laura Ricci, Vittorio Scarano, Miguel A. Vega-Rodríguez, Ana Lucia Varbanescu, Sascha Hunold, Stephen L. Scott, Stefan Lankes, and Josef Weidendorfer, editors, *Euro-Par 2016: Parallel Processing Workshops*, pages 284–295, Cham, 2017. Springer International Publishing.

[52] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. 2, 01 2007.

[53] Floriment Klinaku, Dominik Bilgery, and Steffen Becker. The applicability of palladio for assessing the quality of cloud-based microservice architectures. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ECSA '19, page 34–37, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3344948.3344961.

[54] H. Knoche and W. Hasselbring. Using microservices for legacy software modernization. *IEEE Software*, 35(3):44–49, 2018.

[55] Kenichi Kobayashi, Manabu Kamimura, Koki Kato, Keisuke Yano, and Akihiko Matsuo. Feature-gathering dependency-based software clustering using dedication and modularity. *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, Sep 2012. http://dx.doi.org/10.1109/ICSM.2012.6405308.

[56] A. Koschel, I. Astrova, and J. Dötterl. Making the move to microservice architecture. In *2017 International Conference on Information Society (i-Society)*, pages 74–79, 2017.

[57] Alexander Krause, Christian Zirkelbach, Wilhelm Hasselbring, Stephan Lenga, and Dan Kröger. Microservice decomposition via static and dynamic analysis of the monolith, 2020.

[58] Martin Lehmann and Frode Eika Sandnes. A framework for evaluating continuous microservice delivery strategies. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, New York, NY, USA, 2017. Association for Computing Machinery. https://doi.org/10.1145/3018896.3018961.

[59] Fei Li and Lars Gelbke. Microservice architecture in industrial software delivery on edge devices. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3234152.3234196.

[60] ACM Digital Library. About the acm digital library. https://dl.acm.org/about.

[61] Pingrong Lin, Zheyuan Lin, and Xiaoquan Shi. Research on optimization of course selection system based on micro service and dynamic resource extension. In *Proceedings of the 2019 4th International Conference on Big Data and Computing*, ICBDC 2019, page 115–119, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3335484.3335546.

[62] Huan Liu, Zhiying Cao, and Xiuguo Zhang. An efficient algorithm of context-clustered microservice discovery. In *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*, CSAE '18, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3207677.3277949.

[63] R. J. Machado, J. M. Fernandes, P. Monteiro, and H. Rodrigues. Transformation of uml models for service-oriented software architectures. In *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)*, pages 173–182, 2005.

[64] Prabal Mahanta and Suchin Chouta. Translating a legacy stack to microservices using a modernization facade with performance optimization for container deployments. In Christophe Debruyne, Hervé Panetto, Wided Guédria, Peter Bollen, Ioana Ciuciu, George Karabatis, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems: OTM 2019 Workshops*, pages 143–154, Cham, 2020. Springer International Publishing.

[65] Salvatore Augusto Maisto, Beniamino Di Martino, and Stefania Nacchia. From monolith to cloud architecture using semi-automated microservices modernization. In Leonard Barolli, Peter Hellinckx, and Juggapong Natwichai, editors, *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 638–647, Cham, 2020. Springer International Publishing.

[66] Jeremy M.R. Martin. *Use Case Driven Microservices Architecture Design*, volume 70 of *Concurrent Systems Engineering Series*, page 463–474. IOS Press BV, 2019.

[67] G. Mazlami, J. Cito, and P. Leitner. Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531, 2017.

[68] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar. Microservices: Migration of a mission critical system. *IEEE Transactions on Services Computing*, pages 1–1, 2018.

[69] Nisha Gopinath Menon. Why a monolith was the solution to the microservices problem, Mar 2020. https://www.cognitiveclouds.com/insights/why-a-monolith-was-the-solution-to-the-microservices-problem/.

[70] Archived Docs Microsoft. Chapter 3: Architectural patterns and styles, Jan 2010. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117(v=pandp.10)?redirectedfrom=MSDN.

[71] Azure Microsoft. Create your azure free account today. https://azure.microsoft.com/en-us/.

[72] Ola Mustafa and Jorge Gomez. Optimizing economics of microservices by planning for granularity leve. 04 2017.

[73] Seyyed Yahya Nabavi and Omid Bushehrian. An adaptive plan-oriented and continuous software migration to cloud in dynamic enterprises. *Software: Practice and Experience*, 49(9):1365–1378, 2019. https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2725.

[74] R. Nakazawa, T. Ueda, M. Enoki, and H. Horii. Visualization tool for designing microservices with the monolith-first approach. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 32–42, 2018.

[75] Stephen C. North, Apr 2004. http://graphviz.gitlab.io/pdf/neatoguide.pdf.

[76] Luís Nunes, Nuno Santos, and António Rito Silva. From a monolith to a microservices architecture: An approach based on transactional contexts. In Tomas Bures, Laurence Duchien, and Paola Inverardi, editors, *Software Architecture*, pages 37–52, Cham, 2019. Springer International Publishing.

[77] L. O'Brien, P. Merson, and L. Bass. Quality attributes for service-oriented architectures. In *International Workshop on Systems Development in SOA Environments (SDSOA'07: ICSE Workshops 2007)*, pages 3–3, 2007.

[78] Kasia Pawlaczyk. The pros and cons of rewriting the app from scratch, Sep 2016. https://www.netguru.com/blog/the-pros-and-cons-of-rewriting-the-app-from-scratch, journal=Netguru Blog on Project Management.

[79] K. J. P. G. Perera and I. Perera. A rule-based system for automated generation of serverless-microservices architecture. In *2018 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–8, 2018.

[80] K. J. P. G. Perera and I. Perera. Thearchitect: A serverless-microservices based high-level architecture generation tool. In *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, pages 204–210, 2018.

[81] Tina Poklepović Peričić and Sarah Tanveer. Why systematic reviews matter, Jul 2019. https://www.elsevier.com/connect/authors-update/why-systematic-reviews-matter?aaref=https%3A%2F%2Fwww.google.com%2F.

[82] Ilaria Pigazzini, Francesca Arcelli Fontana, and Andrea Maggioni. Tool support for the migration to microservice architecture: An industrial case study. In Tomas Bures, Laurence Duchien, and Paola Inverardi, editors, *Software Architecture*, pages 247–263, Cham, 2019. Springer International Publishing.

[83] F. Ponce, G. Márquez, and H. Astudillo. Migrating from monolithic architecture to microservices: A rapid review. In *38th International Conference of the Chilean Computer Science Society (SCCC 2019)*, 2019.

[84] F. Ponce, G. Márquez, and H. Astudillo. Migrating from monolithic architecture to microservices: A rapid review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–7, 2019.

[85] Florian Rademacher, Jonas Sorgalla, Sabine Sachweh, and Albert Zündorf. A model-driven workflow for distributed microservice development. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 1260–1262, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3297280.3300182.

[86] Florian Rademacher, Jonas Sorgalla, Philip Nils Wizenty, Sabine Sachweh, and Albert Zündorf. Microservice architecture and model-driven development: Yet singles, soon married (?). In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3234152.3234193.

[87] Zhongshan Ren, Wei Wang, Guoquan Wu, Chushu Gao, Wei Chen, Jun Wei, and Tao Huang. Migrating web applications from monolithic structure to microservices architecture. In *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*, Internetware '18, New York, NY, USA, 2018. Association for Computing Machinery. https://doi.org/10.1145/3275219.3275230.

[88] Kiana Rostami, Johannes Stammel, Robert Heinrich, and Ralf Reussner. Architecture-based assessment and planning of change requests. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, QoSA '15, page 21–30, New York, NY, USA, 2015. Association for Computing Machinery. https://doi.org/10.1145/2737182.2737198.

[89] Peter Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20(1):53–65, November 1987. https://doi.org/10.1016/0377-0427(87)90125-7.

[90] Nuno Santos, Carlos E. Salgado, Francisco Morais, Mónica Melo, Sara Silva, Raquel Martins, Marco Pereira, Helena Rodrigues, Ricardo J. Machado, Nuno Ferreira, and Manuel Pereira. A logical architecture design method for microservices architectures. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ECSA '19, page 145–151, New York, NY, USA, 2019. Association for Computing Machinery. https://doi.org/10.1145/3344948.3344991.

[91] Sarita and S. Sebastian. Transform monolith into microservices using docker. In *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, pages 1–5, 2017.

[92] A. Sayara, M. S. Towhid, and M. S. Hossain. A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling. In *2017 20th International Conference of Computer and Information Technology (ICCIT)*, pages 1–6, 2017.

[93] Anfel Selmadji, Abdelhak-Djamel Seriai, Hinde Lilia Bouziane, Christophe Dony, and Rahina Oumarou Mahamane. Re-architecting oo software into microservices. In Kyriakos Kritikos, Pierluigi Plebani, and Flavio de Paoli, editors, *Service-Oriented and Cloud Computing*, pages 65–73, Cham, 2018. Springer International Publishing.

[94] M. Shahin, M. Ali Babar, and L. Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017.

[95] Heleno Cardoso da Silva Filho and Glauco de Figueiredo Carneiro. Strategies reported in the literature to migrate to microservices based architecture. In Shahram Latifi, editor, *16th International Conference on Information Technology-New Generations (ITNG 2019)*, pages 575–580, Cham, 2019. Springer International Publishing.

[96] Springer. About springer. https://www.springer.com/gp/about-springer.

[97] Chang-ai Sun, Jing Wang, Jing Guo, Zhen Wang, and Li Duan. A reconfigurable microservice-based migration technique for iot systems. In Sami Yangui, Athman Bouguettaya, Xiao Xue, Noura Faci, Walid Gaaloul, Qi Yu, Zhangbing Zhou, Nathalie Hernandez, and Elisa Y. Nakagawa, editors, *Service-Oriented Computing – ICSOC 2019 Workshops*, pages 142–155, Cham, 2020. Springer International Publishing.

[98] D. Taibi, F. Auer, Valentina Lenarduzzi, and M. Felderer. From monolithic systems to microservices: An assessment framework. *ArXiv*, abs/1909.08933, 2019.

[99] D. Taibi, V. Lenarduzzi, and C. Pahl. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5):22–32, 2017.

[100] Davide Taibi and Kari Systä. From monolithic systems to microservices: A decomposition framework based on process mining. 05 2019.

[101] Branko Terzic, Vladimir Dimitrieski, S. Kordic, and I. Lukovic. A model-driven approach to microservice software architecture establishment. In *FedCSIS*, 2018.

[102] S. Tyszberowicz and A. Raman. The easycrc tool. In *2007 International Conference on Software Engineering Advances*, page 52, Los Alamitos, CA, USA, aug 2007. IEEE Computer Society. https://doi.ieeecomputersociety.org/10.1109/ICSEA.2007.72.

[103] Shmuel Tyszberowicz, Robert Heinrich, Bo Liu, and Zhiming Liu. Identifying microservices using functional decomposition. In Xinyu Feng, Markus Müller-Olm, and Zijiang Yang, editors, *Dependable Software Engineering. Theories, Tools, and Applications*, pages 50–65, Cham, 2018. Springer International Publishing.

[104] w3schools. How to - sort a table. https://www.w3schools.com/howto/howto_js_sort_table.asp.

[105] John Wade. Greenfield vs. brownfield software development, Sep 2018. https://synoptek.com/insights/it-blogs/greenfield-vs-brownfield-software-development/.

[106] Philip Wizenty, Jonas Sorgalla, Florian Rademacher, and Sabine Sachweh. Magma: Build management-based generation of microservice infrastructures. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, ECSA '17, page 61–65, New York, NY, USA, 2017. Association for Computing Machinery. https://doi.org/10.1145/3129790.3129821.

[107] K. Yano and A. Matsuo. Data access visualization for legacy application maintenance. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 546–550, 2017.

[108] P. Yugopuspito, F. Panduwinata, and S. Sutrisno. Microservices architecture: Case on the migration of reservation-based parking system. In *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pages 1827–1831, 2017.

[109] P. Yugopuspito, F. Panduwinata, and S. Sutrisno. Microservices architecture: Case on the migration of reservation-based parking system. In *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pages 1827–1831, 2017.

[110] Mikulas Zaymus. *Decomposition of monolithic web application to microservices*. PhD thesis, 2017. https://www.theseus.fi/handle/10024/131110.

[111] Yixue Zhao and Nenad Medvidovic. A microservice architecture for online mobile app optimization. In *Proceedings of the 6th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '19, page 45–49. IEEE Press, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, 01.12.2020, Qiuran

place, date, signature